DTIC FILE COPY

Productivity Engineering in the UNIX† Environment

AD-A229 683

Pan I: An Introduction for Users

Technical Report

DTIC
ELECTE
NOV 29 1990
D

S. L. Graham
Principal Investigator

(415) 642-2059

"The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government."

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

Contract No. N00039-84-C-0089

August 7, 1984 - August 6, 1987

Arpa Order No. 4871

†UNIX is a trademark of AT&T Bell Laboratories

DO NOT REMOVE
ZOAAAAAA8451256O

Pan I
An Introduction For Users[1]

Robert A. Ballance
Michael L. Van De Vanter

DTIC
COPY
INSPECTED
6

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, California 94720

September, 1986
Revised September, 1987

Accesion For	
NTIS CRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

PIPER *Working Paper 87-5*

Abstract

Pan is a prototype and testbed for language-based editors and viewers. Its design addresses the needs of experienced users who manage complex objects such as large software systems. All of *Pan*'s components are multi-lingual, incremental, description-driven, customizable, and extensible. Viewing is facilitated by semantics-based browsing and an object model which integrates text and structure. *Pan* is intended to share information with other tools, allowing integration into a larger language, program, and document development environment.

This document, a users manual, describes the basic operational facilities of *Pan I*, the current implementation. It explains the concepts behind *Pan*'s editing environment, introduces editing commands, and discusses techniques for customization. Appendices list command bindings (to both keystrokes and menus), buffer options, buffer flags, and a compatibility guide for GNU Emacs users. (KR

[1]Sponsored by the Defense Advanced Research Projects Agency (DoD), Arpa Order No. 4871 (monitored by Space and Naval Warfare Systems Command under Contract No. N00039-84-C-0089), by IBM under IBM Research Contract No. 564516, and by the State of California MICRO program. Robert A. Ballance and Michael L. Van De Vanter were supported in part by MICRO fellowships.

APPROVED FOR PUBLIC RELEASE
DISTRIBUTION UNLIMITED

Contents

1 Introduction

Pan is an editor for text- and tree-structured documents that uses the mouse, menus, and multiple windows to provide "cut and paste" editing. The system runs on Sun Workstations[2] under under UNIX[3] and Release 3.2 (and later) of SunView.

The text-oriented facilities of *Pan* are modeled on the *Emacs* family of text editors. Users familiar with an *Emacs*-style editor will have little trouble learning *Pan*. *Pan* is extensible and customizable in the spirit of *Emacs*[6].

Pan also provides for manipulating and editing programs using the syntax and semantics of the language being edited. The current version of *Pan* uses only information about the syntax of the language; the component that uses semantic information is now under development.

This document is an informal introduction to *Pan I*. It contains essential information for editing with *Pan*. To extend the system, though, you'll want the information in *The Pan Extension Manual*[2]. Until that manual is available, you may want to contact the *Pan* group (panpipes@sequoia) directly. For more general background on *Pan*, consult *The Architecture of* Pan *I*[3].

Throughout this document, "*Pan*" refers to *Pan I*, the prototype implementation. *Pan II*, a major revision of *Pan I*, is being designed. Your constructive comments on *Pan I* will help us to provide a better environment with *Pan II*. Let us know what you think!

2 Concepts

This section is a brief introduction to the terminology and notation of *Pan*. Since many of the concepts differ in meaningful ways from similar notions in *Emacs*, it bears careful attention. Appendix G compares the text-oriented concepts and facilities of *Pan* with those supported by *GNU Emacs*[7]

2.1 Buffers

A **buffer** is the focus of editing attention for a single object. (A single object is a text file in the current version.) During editing, *Pan* manages several buffers.

Along with the object being edited, each buffer has

- one or more viewers,
- a single visible selection,
- key, menu, and operand-level bindings,
- an operand-level setting,
- option values,
- flag values,

[2]Sun Workstation, and SunView are registered trademarks of Sun Microsystems, Inc.
[3]UNIX is a registered trademark of AT&T Bell Laboratories.

- a mark stack,
- a kill ring, and
- definitions of character syntax classes.

The viewers associated with a buffer share the selection, bindings, options, and other values owned by that buffer.

All buffers in *Pan* are named. The name of a buffer is the name of the object being edited in that buffer. In the current release, the name is the name of the file being edited.

The base buffer and the help buffer are special buffers maintained by the system. The **base buffer** is your doorway to *Pan*: within its edit window are the names of the buffers being edited. As new buffers are created, their names are added to this list.

The base buffer differs from other buffers in two ways: only the base buffer contains `Quit` in its frame menu and only the base buffer can be made iconic. The base buffer is *always* visible on the screen, perhaps as an icon.

The help buffer is used for displaying help and other information. See Section 3.6 for more about help facilities.

Most commands act on the **active viewer** of the **active buffer**. The active viewer is the viewer in which the most recent keystroke or mouse action occurred. The active buffer is usually the buffer that owns the active viewer.

Figure 1 shows the screen of a workstation running *Pan*. The base buffer appears in the upper left corner; it lists two other buffers, the help buffer and a buffer named "`manual.tex`". Buffer `manual.tex` (containing a file of the same name) is currently active and has two separate viewers open onto it. The help buffer, visible in the upper right corner, currently shows the bindings in effect for the active buffer.

2.2 Viewers

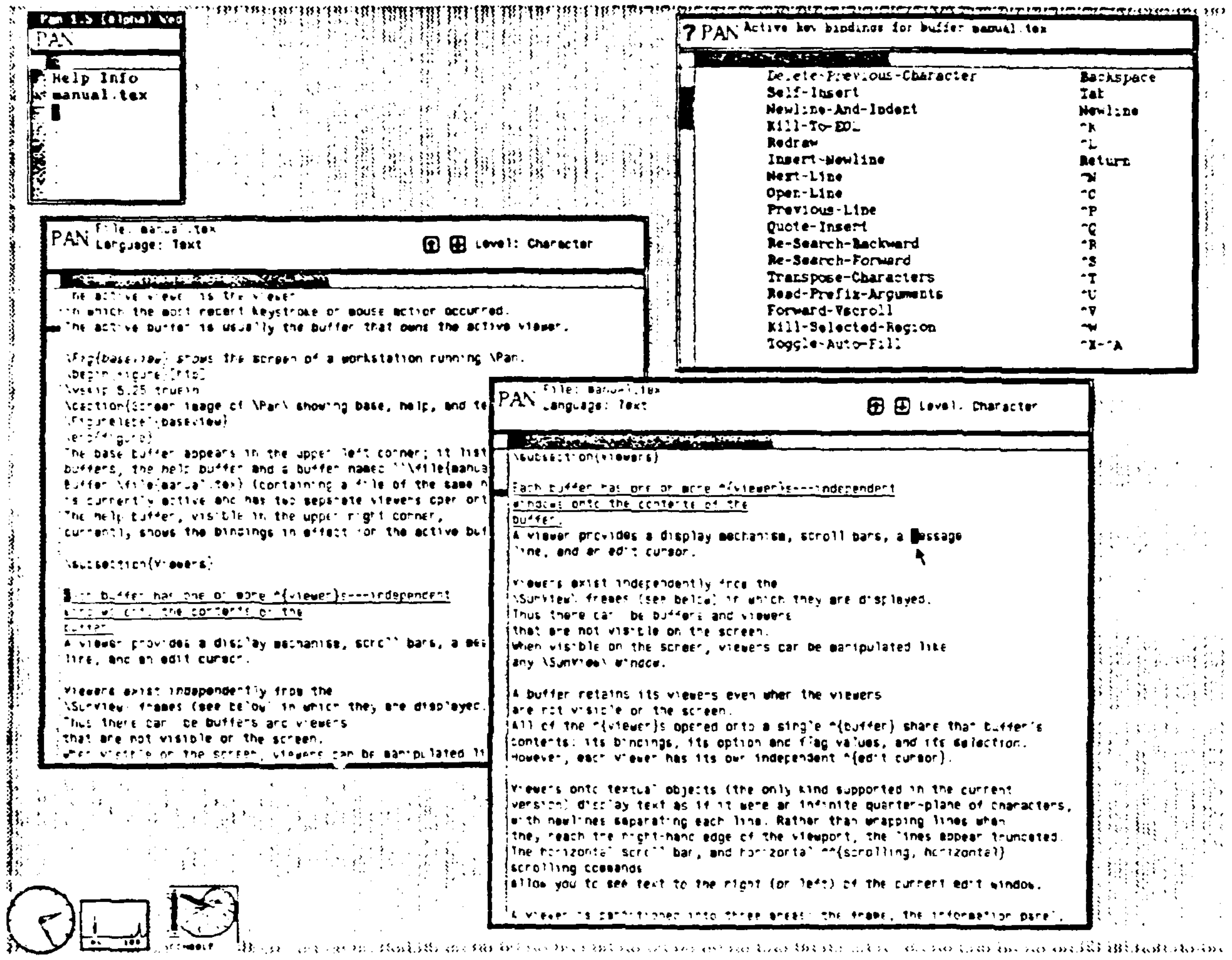
Each buffer has one or more viewers—independent windows onto the contents of the buffer. A viewer provides a display mechanism, scroll bars, a message line, and an edit cursor.

Viewers exist independently from the SunView frames (see below) in which they are displayed. Thus there can be buffers and viewers that are not visible on the screen. When visible on the screen, viewers can be manipulated like any SunView window.

A buffer retains its viewers even when the viewers are not visible on the screen. All of the viewers opened onto a single buffer share that buffer's contents: its bindings, its option and flag values, and its selection. However, each viewer has its own independent edit cursor.

Viewers onto textual objects (the only kind supported in the current version) display text as if it were an infinite quarter-plane of characters, with newlines separating each line. Rather than wrapping lines when they reach the right-hand edge of the viewport, the lines appear truncated. The horizontal scroll bar, and horizontal scrolling commands allow you to see text to the right (or left) of the current edit window.

A viewer is partitioned into three areas: the frame, the information panel, and the editing area.



```
Pan 1.5 (Alpha) Ved
PAN
Help Info
manual.tex
```

```
PAN File: manual.tex
    Language: Text                                   Level: Character

The active viewer is the viewer
in which the most recent keystroke or mouse action occurred.
The active buffer is usually the buffer that owns the active viewer.

\Fig{baseview} shows the screen of a workstation running \Pan.
\begin{figure}[htb]
\vskip 5.25 truein
\caption{Screen image of \Pan\ showing base, help, and te
\figurelabel{baseview}
\end{figure}
The base buffer appears in the upper left corner; it list
buffers, the help buffer and a buffer named ``\file{manua
Buffer \file{manual.tex} (containing a file of the same n
is currently active and has two separate viewers open ont
The help buffer, visible in the upper right corner,
currently shows the bindings in effect for the active buf

\subsection{Viewers}

Each buffer has one or more ^{viewer}s---independent
windows onto the contents of the
buffer.
A viewer provides a display mechanism, scroll bars, a mes
line, and an edit cursor.

Viewers exist independently from the
\SunView\ frames (see below) in which they are displayed.
Thus there can be buffers and viewers
that are not visible on the screen.
When visible on the screen, viewers can be manipulated li
```

```
? PAN Active key bindings for buffer manual.tex

Delete-Previous-Character          Backspace
Self-Insert                        Tab
Newline-And-Indent                 Newline
Kill-To-EOL                        ^K
Redraw                             ^L
Insert-Newline                     Return
Next-Line                          ^N
Open-Line                          ^O
Previous-Line                      ^P
Quote-Insert                       ^Q
Re-Search-Backward                 ^R
Re-Search-Forward                  ^S
Transpose-Characters               ^T
Read-Prefix-Arguments              ^U
Forward-Vscroll                    ^V
Kill-Selected-Region               ^W
Toggle-Auto-Fill                   ^X-^A
```

```
PAN File: manual.tex
    Language: Text                                   Level: Character

\subsection{viewers}

Each buffer has one or more ^{viewer}s---independent
windows onto the contents of the
buffer.
A viewer provides a display mechanism, scroll bars, a message
line, and an edit cursor.

Viewers exist independently from the
\SunView\ frames (see below) in which they are displayed.
Thus there can be buffers and viewers
that are not visible on the screen.
When visible on the screen, viewers can be manipulated like
any \SunView\ window.

A buffer retains its viewers even when the viewers
are not visible on the screen.
All of the ^{viewer}s opened onto a single ^{buffer} share that buffer's
contents: its bindings, its option and flag values, and its selection.
However, each viewer has its own independent ^{edit cursor}.

Viewers onto textual objects (the only kind supported in the current
version) display text as if it were an infinite quarter-plane of characters,
with newlines separating each line. Rather than wrapping lines when
they reach the right-hand edge of the viewport, the lines appear truncated.
The horizontal scroll bar, and horizontal ^^{scrolling, horizontal}
scrolling commands
allow you to see text to the right (or left) of the current edit window.

A viewer is partitioned into three areas: the frame, the information panel,
```

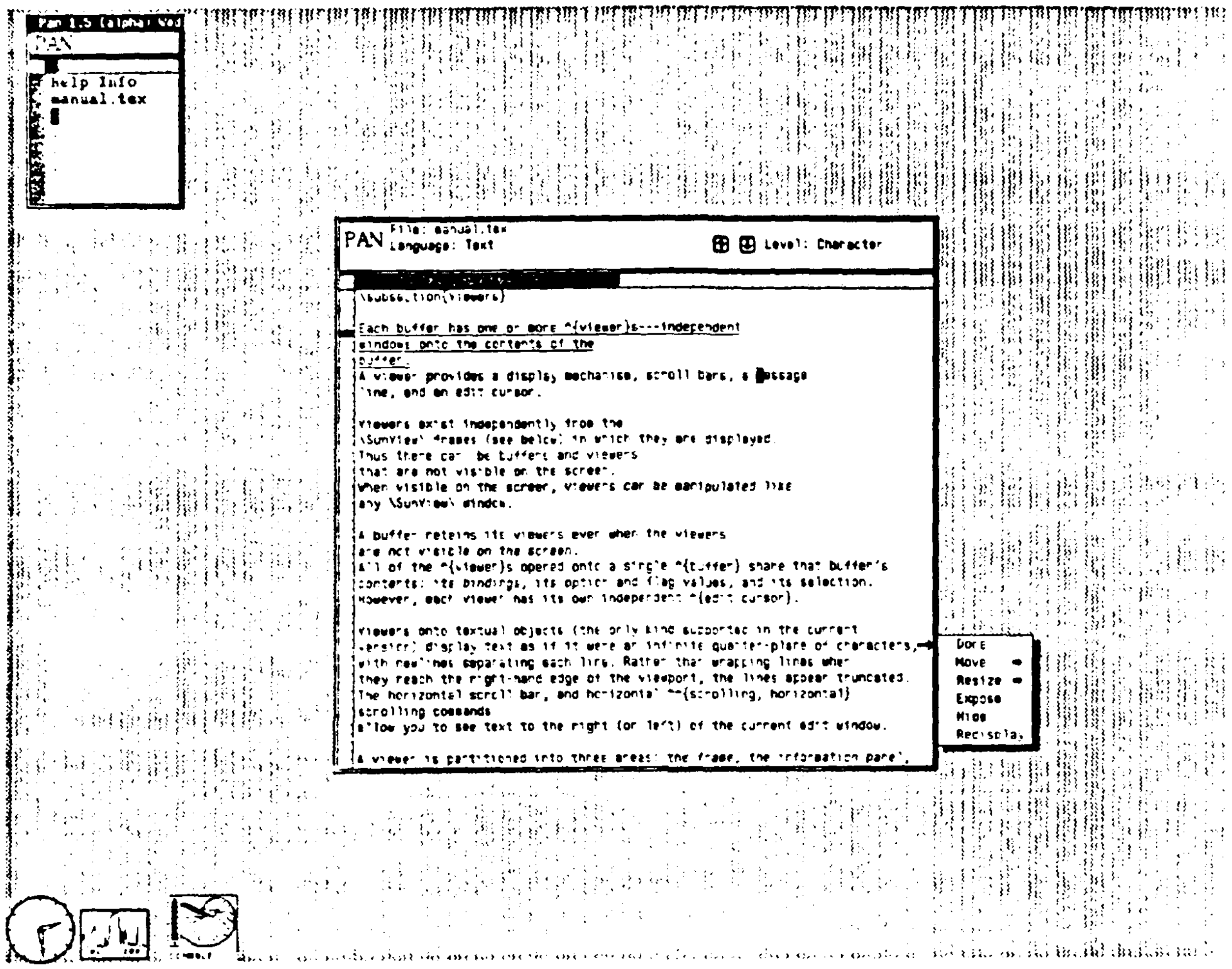
Figure 1: Screen image of *Pan* showing base, help, and text buffers

2.2.1 Frames

The frame surrounding a viewer responds to the normal SunView protocols[1]. Those protocols can be used to independently position and size the viewer. Figure 2 shows a viewer with its frame menu exposed.

The frame associated with the base buffer is the only one containing the items `Close` and `Quit` in its frame menu. When `Close` is chosen, all of the visible viewers disappear from the screen, and an icon for *Pan* appears. Opening the icon reopens the other viewers. When `Quit` is chosen, you will be offered the chance to save any modified buffers before exiting. This is the normal way to terminate an editing session.

Frames other than the frame associated with the base buffer have the `Done` menu item in their frame menu. Selecting this item causes its viewer to disappear from the screen. Internal state of the viewer is retained even when the viewer is not visible.



```
File: manual.tex
Language: Text

\subsection{viewers}

Each buffer has one or more ^{viewer}s---independent
windows onto the contents of the
buffer.
A viewer provides a display mechanism, scroll bars, a message
line, and an edit cursor.

Viewers exist independently from the
\SunView\ frames (see below) in which they are displayed.
Thus there can be buffers and viewers
that are not visible on the screen.
When visible on the screen, viewers can be manipulated like
any \SunView\ window.

A buffer retains its viewers even when the viewers
are not visible on the screen.
All of the ^{viewer}s opened onto a single ^{buffer} share that buffer's
contents: its bindings, its option and flag values, and its selection.
However, each viewer has its own independent ^{edit cursor}.

Viewers onto textual objects (the only kind supported in the current
version) display text as if it were an infinite quarter-plane of characters,
with newlines separating each line. Rather than wrapping lines when
they reach the right-hand edge of the viewport, the lines appear truncated.
The horizontal scroll bar, and horizontal ^^{scrolling, horizontal}
scrolling commands
allow you to see text to the right (or left) of the current edit window.

A viewer is partitioned into three areas: the frame, the information panel,
```

Figure 2: *Pan* with a frame menu visible

Due to limitations of SunView, only six frames are available to *Pan*. This means that you can only have six *Pan* viewers (including the base buffer) visible at any time. Frames are allocated to a viewer when the viewer is opened, and are deallocated when the viewer is closed.

2.2.2 The Information Panel

Inside the frame, at the top of the viewer, is a panel containing information about the object being viewed. This includes the buffer name, the language being edited, the operand selection level, a message line, and the values of various flags. Viewers onto the help buffer contain only a title and a message line.

2.2.3 The Editing Area

Below the information panel is the editing area. This area includes horizontal and vertical scroll bars, and the actual edit window. Pressing the right button of the mouse over the edit window of the viewer activates the *Pan* menus for that buffer. Selecting an item from those menus executes the command bound to that selection.

2.2.4 Font Maps

Every character in *Pan*'s internal text representation contains a font code. Each viewer has an associated **font map**, which associates font codes with internal font descriptors. A font map contains from 1 to 16 entries.

Internally, fonts are referred to as "font 0", "font 1", etc. The first element of the list (font 0) is the **default font**; unspecified font codes revert to the default font.

Pan maintains three default font maps: one for normal viewers, one for the base buffer, and one for the help buffer. You can alter these defaults by setting the options :text-font-map, :base-frame-font-map, and :help-frame-font-map respectively. The standard specification for a font map is a zero-indexed list of font names containing from 1 to 16 names. For example, the default value of the option :text-font-map is

```
("screen.r.12" "screen.b.12" "serif.r.12" "cour.r.12" "cour.b.12")
```

Currently, *Pan* offers only limited support for manipulating font codes in the text representation. Standard text viewers use only the default font. Section 3.12.4 describes a more elaborate use of font maps in conjunction with tree-structured documents.

2.3 Bindings

A **binding** associates a sequence of keyboard or mouse actions with a command. **Key bindings** associate keystroke sequences with commands; **menu bindings** associate menu item selections with commands. **Operand level bindings** associate generic operations, such as "next" or "delete" with operands designated by the current operand level (section 2.3.3).

Key and menu bindings are either *local* to a buffer or are *global* to all buffers. Every buffer may have its own set of key bindings, and even its own menus and menu selections. Naturally, local bindings take precedence over global bindings. Operand level bindings are local to each buffer: there are no global defaults for them.

2.3.1 Key Bindings

Pan, like *Emacs*, provides a live keyboard. Keystrokes (including mouse buttons) are read until a valid binding is detected. When a binding is detected, the associated command is executed.

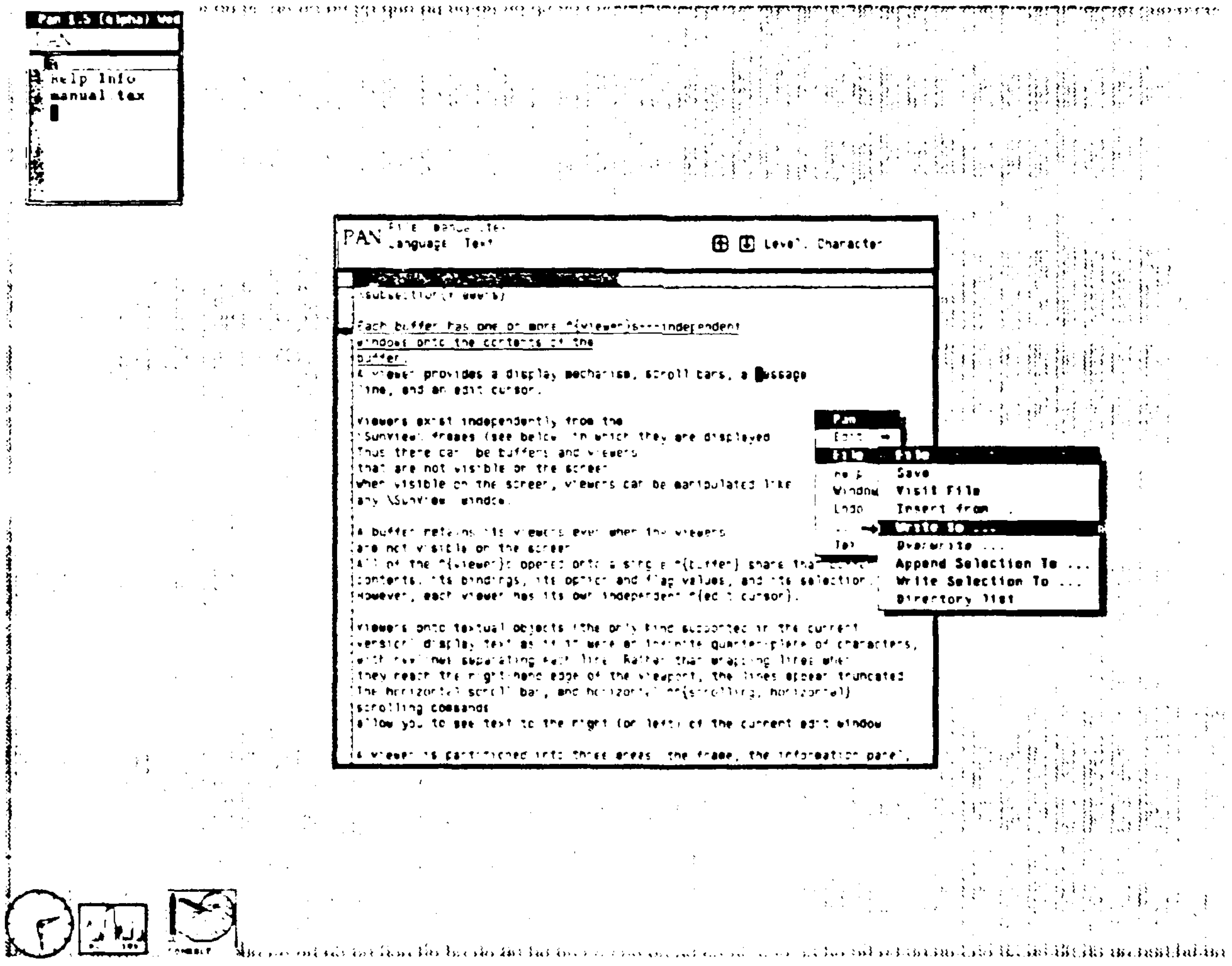
A key binding associates a 1- or 2-character keystroke sequence with a command. The keys `Escape`, `^X`, `^C`, and `^Z` are reserved to be prefix keys in two-keystroke bindings. The `Shift` and `Control` keys are modifier keys rather than prefix keys.

Function keys and mouse buttons can be mapped just like the standard keyboard keys. In fact, it is the standard binding of the right-most mouse button to Execute-From-Menu that implements *Pan*'s menu selection service.

The default bindings for the left-hand function keys (L1–L10) reflect standard SunView usage when appropriate. The default bindings for mouse buttons are similar to the SunView bindings. Appendix A lists *Pan*'s default set of key bindings.

2.3.2 Menu Bindings

A menu binding consists of a a menu, a menu item, and a command name. Like key bindings, menus and menu bindings may be either local or global. The menus associated with a buffer appear when right button of the mouse is pressed while the mouse is positioned over the edit window. Figure 3 shows a menu selection being made.



```
\subsection{Viewers}

Each buffer has one or more ^{viewer}s---independent
windows onto the contents of the
buffer.
A viewer provides a display mechanism, scroll bars, a message
line, and an edit cursor.

Viewers exist independently from the
\SunView frames (see below) in which they are displayed
Thus there can be buffers and viewers
that are not visible on the screen
When visible on the screen, viewers can be manipulated like
any \SunView window.

A buffer retains its viewers even when the viewers
are not visible on the screen
All of the ^{viewer}s opened onto a single ^{buffer} share that buffer's
contents, its bindings, its option and flag values, and its selection.
However, each viewer has its own independent ^{edit cursor}.

Viewers onto textual objects (the only kind supported in the current
version) display text as if it were an infinite quarter-plane of characters,
with newlines separating each line. Rather than wrapping lines when
they reach the right-hand edge of the viewport, the lines appear truncated
The horizontal scroll bar, and horizontal ^{scrolling, horizontal}
scrolling commands
allow you to see text to the right (or left) of the current edit window

A viewer is partitioned into three areas: the frame, the information panel,
```

Figure 3: *Pan* with a buffer menu visible

A menu binding is designated by a menu title and a selection name, denoted typographically by MenuTitle: `SelectionName`. It is not necessary for the selection name to be identical to the name of the command bound to the selection.

Default menu bindings are best discovered by mousing around. Appendices B–D list the default menu bindings for a text buffer, for the base buffer, and for the help buffer.

2.3.3 Operand Level Bindings

A keystroke sequence or menu item can be bound to a command that implements a generic operation. The generic operation, in turn, consults the operand-level bindings and the current operand level of the active viewer to determine the actual editor command to execute.

Each viewer has a **current operand level** which can be used to control the actions of operand-generic operations. The generic operations are: `:next`, `:previous`, `:select`, `:mouse-select`, `:mouse-extend`, `:cut`, `:copy`, `:paste`, and `:delete`.

For example, the key "`^N`" is bound to the command Next-@Level. If the current operand level is "Character", and the binding of the generic command :next at the level "Character" is the command Next-Character, then Next-@Level will execute Next-Character.

Setting the operand level is a bit like changing modes in a moded editor. The current operand level persists across operations. In pure text editing, the operand level is of limited usefulness. When editing objects that have a richer operand domain, such as computer programs, the ability to select and navigate using the operand level bindings is a bonus. For instance, a programming language might define operand levels such as "Expression", "Statement", and "Declaration". Figure 5 on page 25 shows the operand-level choices for a language-based viewer.

The current operand level affects only those commands that consult it. These commands are syntactically distinguished by containing the phrase "@Level" in their name.

2.4 The Edit Cursor

Each viewer has a single **edit cursor**; buffers have as many cursors as they have viewers. The edit cursors are independent from each other. An edit cursor appears on the screen as an inverse-video or outlined box highlighting the character selected by that cursor.

All insertions and deletions occur at the position to the left of the character selected by the cursor. If a command alters text not located at the edit cursor (for instance, by deleting the current selection), the cursor is moved to the point of change. Operations that modify the object being edited move the edit window so that the change (and therefore the cursor) is visible.

A viewer's edit cursor may not be visible on the screen due to scrolling or other motion. The command Frame-Edit-Cursor in the Window menu moves the edit window so that the cursor is visible.

2.5 Regions and The Selection

Many of the text-oriented commands in *Pan* operate on a contiguous sequence of characters called a **region** of text.

Every buffer can have a specially designated region of text called the the **selected region** or simply the **selection**. When the selection is set, it is highlighted in all of the viewers in which it is visible. Figure 1 on page 3 shows a selection shared by two viewers. The value of the option :region-highlight-op determines whether the current selection is highlighted using underlining or inverse-video. Commands that alter the contents of the buffer deselect the current selection.

Pan also has an implicit selection, namely the region between the edit cursor and the top mark of the mark stack. This selection is different for each viewer, since each viewer has a different edit cursor. In *Emacs*, the implicit selection is the only region available, while in *Pan*, it is secondary to the visible selection. Commands that operate on the implicit selection are provided mostly for *Emacs* compatibility.

Pan's style of selection is similar to, but not the same as, SunView's. Clicking the middle button of the mouse selects and highlights the region between the cursor and the mouse, but placing or dragging the cursor does *not* affect the selection. The selection, the position of the mouse, and the position of the edit cursor are all independent.

2.6 Rings

A ring is a circular bounded stack. Adding an item to a ring pushes the other items just like a bounded stack. The oldest value in the stack may be discarded to preserve the boundedness.

Rings can also be "cycled", where the top value is moved to the position of the oldest value, and all of the other values move up—the second youngest becoming the top. Cycling a ring by n values moves the nth element (modulo the size of the ring) to the top. The top value in a ring is called the "contents" of the ring.

Both the clipboard and the kill rings are implemented as rings.

2.6.1 The Clipboard

The **clipboard**—a holder for regions of text—is shared among buffers. A selection can be copied (or cut) to the clipboard and then pasted into another buffer. These operations are modeled on the Macintosh[4] user interface. Unlike the Macintosh clipboard, *Pan*'s clipboard is a ring that contains several items. The size of the clipboard is determined by the value of the option :clipboard-max-size.

2.6.2 Kill Rings

The **kill ring** is a a repository for deleted text. Commands that "kill" text place the killed text into the kill ring. This text can be retrieved at a later time.

The kill ring facility is less powerful than *Emacs*'. In *Pan*, the kill ring is local to a buffer rather than global to all buffers. We are still experimenting with this aspect of the user interface. The size of the kill ring is determined by the value of the option :killring-max-size.

Cutting text to the clipboard does not affect the entry in the kill ring; killing text does not affect the clipboard.

2.7 Marks and the Mark Stack

A **mark** is a character position in a textual object. Marks associate with the character to the left of the position; when text is deleted, affected marks migrate to the beginning of the deletion.

Each buffer has a stack of marks called the **mark stack**. Marks in *Pan* are used for two purposes: to remember a cursor position, and to construct regions. The top mark on the stack is usually referred to as "the mark".

[4] Macintosh is a registered trademark of Apple Computer, Inc.

2.8 Syntax Classes

Every buffer has a set of syntax class definitions. Within a buffer, each ASCII character is a member of exactly one syntax class. The possible classes are: :space-char, :word-char, :punct-char, :lbracket, :rbracket, or :other. Every character in the classes :lbracket or :rbracket must have a matching bracket specified.

2.9 Options, Flags, and Variables

Options, flags, and variables are provided for controlling and extending the system. The following paragraphs provide a brief summary of their properties and use.

Options are used to control user-configurable settings. They are strongly-typed, scoped variables. The value of an option local to a buffer generally hides the value of that option in the global space. Appendix E lists the basic options, their types, and their default values. The value of an option is obtained using the command Option-Value (see section 4.3).

A **flag** is strongly-typed, scoped variable capable of holding only a single bit of information. Like options, flags are scoped relative to buffers. Unlike options, flag values can be displayed on the information panel. For instance, if the :text-modified flag is set (meaning that the text of an object has been modified since its last save), a "*" appears on the information panel. Appendix F lists the default set of flags, together with their display properties.

A *Pan* **variable** is a *Lisp* global variable that is known to the help system.

2.10 Communication with Other SunView Clients

Communication with other workstation client programs using the SunView selection mechanism is not yet implemented.

2.11 What's Where?

Just as a buffer ties several viewers together, there is some global state shared by all buffers. This state includes default bindings, default option values, the list of buffers appearing in the base buffer, and the clipboard.

The following table provides a synopsis of which objects have custody of various other objects. In general, local values override more global values when there is a choice.

	Viewer	Buffer	Global
Buffer List			√
Clipboard			√
Edit Cursor	√		
Flags		√	
Key Bindings		√	√
Kill Ring		√	
Mark Stack		√	
Menu Bindings		√	√
Operand Level	√		
Option Values		√	√
Selection		√	
Syntax Classes		√	
Viewers		√	

3 Editing with *Pan*

This section is an introduction simple editing using *Pan*. At the beginning of each subsection is a list of the commands discussed, together with their default bindings. Control keys are represented by prepending the character "^" in front of the key, e.g., `Control-X` is shown as `^X`, and the keystroke sequence `Control-X Control-F` is shown as `^X-^F` where the hyphen separates the keys in the sequence. The prefix `Escape` is denoted `Esc-`.

Menu bindings are denoted by MenuTitle: `SelectionName` where MenuTitle is the title of the menu and `SelectionName` is the name appearing in the menu. Operand-level bindings are denoted by ⟨Command,"level"⟩.

3.1 Getting Started

Pan for Sun Model 3 workstations resides in `~piper/bin/pan3`. *Pan* can be run from a shell tool, a command tool, or a menu under SunView.

When *Pan* is started, the files named on the command line are read into edit buffers and prepared for editing. Once initialization is complete, the base frame will appear. Section 4.1 provides more information about *Pan*'s start-up processing.

3.2 Quitting *Pan*

Write-Files-Exit	`^X-^C`
Write-Files-Exit	Base Buffer: `Quit`
Exit!	Not bound

Write-Files-Exit is the normal method for terminating an editing session. When modified buffers exist you will be asked whether they should be saved. The command Exit! terminates an editing session without saving any modified buffers.

3.3 Suppling Arguments to Commands

There are three ways to provide arguments to *Pan* commands: by a numeric prefix argument, by setting the current selection, or by responding to a prompt. The actual method used depends on the particular command. Commands that can use a numeric argument normally check for the presence of a numeric prefix, while commands that require textual arguments must use prompting. A few, like Visit-Selection, check for a selection before prompting.

3.3.1 Numeric Prefix Arguments

Read-Prefix-Arguments	^U

The behavior of many commands can be altered by suppling numeric prefix arguments. In most cases, the argument is interpreted as a repetition factor, and the effect of the command is simply repeated. Read-Prefix-Arguments reads the prefix arguments from the keyboard. There are two ways to type such arguments: by typing zero or more minus signs (hyphens) followed by a sequence of digits, or by repeating a keystroke sequence bound to Read-Prefix-Arguments. In the latter case, each repetition corresponds to addition by 4.

3.3.2 Prompts and Pop-Ups

Pan prompts for input by making a small **pop-up window** appear on the screen. The pop-up remains on the screen until you complete the input. *When a pop-up is present it seizes all window input and output on your workstation.*

Pop-ups have one or more "buttons" on their lower edge; clicking the left button of the mouse over one of those areas completes and confirms the prompt. When typing a textual argument into a pop-up, your standard UNIX editing characters serve to edit the input. Pop-ups for textual arguments are confirmed by selecting a button or by hitting the Return key.

3.4 Aborting Commands

Abort-Command	^G
Abort-Command	^C-^G
Abort-Command	Esc-^G
Abort-Command	^X-^G
Abort-Command	^Z-^G
Cancel-Command	"Cancel" on Popups

To abort a keystroke sequence, type ^G with any prefix. To abort a menu selection, move the mouse cursor outside (away from) the menu, and release the mouse button. When responding to a prompt, selecting the Cancel button aborts the command.

Once a command is initiated, there is little you can do to stop it. If the command takes more than an instant to complete, the mouse cursor image, normally an arrow, may be replaced by a light bulb.

3.5 Undoing Actions

Undo	^X-u
Undo	L4 (Undo)
Undo	Pan: Undo

The undo facilities of *Pan* allow you to undo the most recent action or series of similar actions. For instance, typing a series of characters and then invoking Undo will remove the entire series of characters just typed. Undoing a second time restores the text removed by the first Undo.

3.6 Getting Help

Reset-Help-Buffer	Help: Reset

Pan is largely self-documenting. Each viewer has an associated Help menu that provides access to the help information supplied by the system. This information will be displayed in the **help buffer**—a special buffer known to the system. Figure 1 on page 3 shows the help buffer in the upper right-hand corner of the screen. The help buffer acts like a normal text viewer, except that it is altered only by the help commands.

Most help commands delete the text in the help buffer before adding their contribution. Others, such as Describe-Selection and Apropos-Selection add additional information to the buffer. The information in the help buffer can be saved to a text file at any time. The command Reset-Help-Buffer empties the help buffer.

Apropos-Selection	Help: Apropos
Apropos-All-Symbols	Help: Apropos All
Apropos-Symbol	Not bound

The "Apropos" family of commands associate a keyword with a list of command names. For instance, invoking the command Apropos-Symbol and supplying the argument "clipboard" lists, in the help buffer, all of the commands that use the clipboard. In another example, if the text "command" is selected, then invoking Apropos-Selection will list all of the editor commands defined in *Pan*. Both Apropos-Symbol and Apropos-Selection operate identically after their argument has been specified.

Describe-Selection	Help: Describe
Describe-Symbol	Not bound
List-All-Commands	Help: Commands
List-All-Flags	Not bound
List-All-Options	Not bound
List-All-Variables	Not bound
Show-Buffers	^X-^B
Show-Buffers	Help: Buffers
Show-Flag-Values	Help: Flag Values

Show-Option-Values	Help: Option Values
Show-Key-Bindings	Help: Key Bindings
Show-Menu-Bindings	Help: Menu Bindings
Show-Operand-Bindings	Help: Operand Bindings
Describe-Operand-Hierarchy	Help: Operand Hierarchy

Commands beginning with "Describe-" provide detailed information about a *Pan* object. Commands beginning with "List-" simply list the names of all objects of a given type (e.g., all commands or all options). The "Show-" commands provide the detailed information of a "Describe-" command for all objects of a given type.

3.7 Buffers and Viewers

The commands presented in this section describe basic buffer and viewer handling.

3.7.1 Visiting Files and Buffers

Visit-Selection	Base Buffer: Visit
Visit-Selection	File: Visit File
Visit-File	^X-^F
Visit-File	Base Buffer: Visit File
Visit-File	File: Visit File
Visit-Buffer	^X-b
Remove-Buffer	^X-k
Remove-Selected-Buffer	Base Buffer: Remove Selected Buffer
List-Files	^X-^D
List-Files	Base Buffer: Directory Listing
Show-Status-Line	^X-=

To edit an object, one must first have opened a viewer onto it. The Visit-Selection command is one way to open a viewer. This command normally appears as `Visit` in either the top-level or the File menus. It is used by first selecting a buffer name in the base buffer.

Visit-File prompts for a file name. If that file is not already in a buffer, a new buffer is allocated and the file is read and prepared for editing.

The "Remove-" commands remove a given buffer from the set of buffers. If the object has been modified since it was last saved, you will be offered the chance to save the buffer. Once Remove-Buffer or Remove-Selected-Buffer has been executed, all of the state associated with the removed buffer is lost.

List-Files prompts for a directory or file name expression, and lists those files, using the *ls* command, in the help buffer. Flags to the *ls* command are determined by the value of the option :ls-flags.

The command Show-Status-Line can be used to display status information about the active buffer in the information panel. The option :mode-line-fmt specifies which status information is displayed.

3.7.2 Saving and Writing Files

Save-Buffer-File	`^X-^S`
Save-Buffer-File	File: `Save`
Save-All-Buffers	`^X-Return`
Write-Files-Exit	`^X-^C`
Write-To-File	`^X-^W`
Write-To-File	File: `Write To ...`
Overwrite-File!	File: `Overwrite ...`
Write-Selection-To-File	File: `Write Selection To ...`
Append-Selection-To-File	File: `Append Selection To ...`
Toggle-Read-Only	`^X-^Q`

Save-Buffer-File saves the contents of the current buffer in its associated file; Save-All-Buffers saves all of the modified buffers being edited. Write-Files-Exit performs the standard termination sequence of saving modified buffers and exiting. Both Save-All-Buffers and Write-Files-Exit prompt for whether a particular buffer is to be written to file.

All of the commands Write-To-File, Write-Selection-To-File, Overwrite-File!, and Append-Selection-To-File prompt for the name of a file to write.

Toggle-Read-Only toggles the :read-only flag on the buffer. When this flag is set, a "$"appears on the information panel and you will be prevented from modifying or writing that file. This flag is set by default when you commence editing a file for which you do not have permission to write. Toggling the :read-only flag does not affect the permissions on the file.

3.7.3 Manipulating Viewers

Open-Another-Viewer	`^X-2`
Open-Another-Viewer	Window: `Open Another Viewer`
Close-Active-Viewer	`^X-0`
Close-Active-Viewer	`L7 (Open)`
Close-Active-Viewer	Window: `Close`
Close-Active-Viewer	SunView Frame: `Done`
Redraw	`^L`

When you visit a buffer that has no visible viewers, *Pan* will reopen the buffer's most recently closed viewer. If the buffer has no viewers, a new viewer is created. To open a second (or third, or fourth, ...) viewer onto a buffer, execute the command Open-Another-Viewer.

Closing a viewer causes it to disappear from the screen. Its internal state is retained so that the viewer can be reopened later. To close a viewer, execute Close-Active-Viewer in the viewer that you wish to close. Alternatively, press the key L7 (Open) when the viewer that you wish to close has the input focus. Closing the viewer associated with the base buffer causes the base buffer to become iconic, and closes all other visible viewers.

Viewers are reopened using the same commands as are used for opening new viewers. They are reopened in last-in, first-out order relative to the order in which they were closed. Closing a viewer frees its associated viewport for use by other viewers.

Redraw redraws the active edit window.

3.8 Scrolling

Pan's scrolling behavior is a simplified version of the SunView scrolling protocols. The scroll bars in a viewer respond to simple scrolling commands. For vertical scrolling, pressing the left button in the vertical scroll bar moves the edit window toward the end of the file, the right button moves the edit window towards the beginning of the file, and the middle button thumbs the edit window to the point in the object indicated by the scroll cursor.

When scrolling horizontally, pressing the left button in the horizontal scroll bar moves the edit window toward the end of the line, the right button moves the edit window towards the beginning of the line, and the middle button thumbs the edit window to the point in the line indicated by the scroll cursor.

Mouse-Forward-Vscroll	`(Scrollbar) Mouse_Left`
Mouse-Backward-Vscroll	`(Scrollbar) Mouse_Right`
Mouse-Abs-Vscroll	`(Scrollbar) Mouse_Middle`
Forward-Vscroll	`^V`
Backward-Vscroll	`Esc-v`
Left-Hscroll	`^X-<`
Right-Hscroll	`^X->`
Frame-Edit-Cursor	Window: `Frame Edit Cursor`

The commands Mouse-Forward-Vscroll, Mouse-Backward-Vscroll, and Mouse-Abs-Vscroll are bound to the left-, right-, and middle mouse buttons when the mouse is over the vertical scroll bar.

During vertical scrolling, when the option :proportional-scroll is set to be true (in *Lisp*, `'t`), the amount that the edit window is scrolled depends upon the distance between the mouse cursor and the top of the scroll bar. For small movements, place the cursor near the top of the scroll bar. For larger movements, place the mouse cursor near the bottom of the scroll bar. To scroll an entire screen, place the mouse cursor opposite to the last line of text visible in the edit window.

When the option :proportional-scroll is set to be false (in *Lisp*, `'nil`), the vertical scrolling commands move the edit window by a full screen at a time.

To "thumb" the viewer to an absolute position in the file, place the mouse cursor into the scroll bar and press `Mouse_Middle`. The viewer will be moved to the line in the file corresponding relative distance between the top of the scroll bar and the position of the mouse cursor.

Scrolling the screen by a full screen at a time can also be achieved by using Forward-Vscroll and Backward-Vscroll. Both of those commands consult the numeric prefix argument to determine the number of screens to move.

Left-Hscroll and Right-Hscroll are keyboard variants of the horizontal scrolling commands.

The command Frame-Edit-Cursor shifts the edit window so that the edit cursor will be visible.

3.9 Cursor Motion

Mouse-Select-@Level	Mouse_Left
Cursor-To-Mouse	⟨Select,"Character"⟩
Next-@Level	^F
Next-Character	⟨Next,"Character"⟩
Next-Word	Esc-f
Next-Word	⟨Next,"Word"⟩
Next-Line	^N
Next-Line	⟨Next,"Line"⟩
Previous-@Level	^B
Previous-Character	⟨Previous,"Character"⟩
Previous-Character	^H (Backspace)
Previous-Word	Esc-b
Previous-Word	⟨Previous,"Word"⟩
Previous-Line	^P
Previous-Line	⟨Previous,"Line"⟩
Move-To-BOL	^A
Move-To-EOL	^E
Move-To-BOB	Esc-<
Move-To-EOB	Esc->
End-Of-Word	Not bound
First-Non-Blank	Esc-m
Goto-Line	^X-1

The left button of the mouse is bound to the command Mouse-Select-@Level. When the operand level is set to "Character", it sets the edit cursor to the character selected by the mouse icon. The cursor can also be moved using cursor motion commands shown above. The "Next-" and "Previous-" commands use the numeric prefix argument to determine the number of units to move.

Move-To-BOL moves the edit cursor to the first position on the current line; Move-To-EOL moves the edit cursor last position on the current line. To move to the beginning or end of the object being edited, use Move-To-BOB or Move-To-EOB, respectively

End-Of-Word moves the cursor to the end of the word that encloses the cursor. Finally, First-Non-Blank moves the cursor to the first character (that is not white space) on the current line.

The command Goto-Line moves the cursor to the line specified by the numeric prefix argument. If there is no prefix argument, Goto-Line prompts for a line number. Internally, *Pan* treats the first line of a file as line number 0. If the option :zero-index-lines is false, the argument to Goto-Line is treated as a 1-indexed line number and is converted appropriately.

3.9.1 Mark Commands

Set-Mark	^@
Pop-Mark	Not bound

Swap-Dot-And-Mark	^X-^X
Dot-To-Mark	Not bound

Set-Mark and Push-Mark push and pop the mark stack of the active buffer. The commands Swap-Dot-And-Mark and Dot-To-Mark move the edit cursor to the position indicated by the top of the mark stack. They differ in that the first exchanges the cursor's position with the top mark, while the second pops stack.

3.10 Editing Text

This section describes the commands for text manipulation.

3.10.1 Setting the Operand Level

Up-@Level	F2
Up-@Level	Panel Item
Down-@Level	F3
Down-@Level	Panel Item
Set-@Level	Panel Menu Item
Set-@Level-To-Character	F1

You can ignore the operand level settings, and *Pan* will operate pretty much like *Emacs*. If you want to experiment, however, you will notice that each viewer panel contains a sequence of items labeled "Level:". The small button containing an upward-pointing arrow is bound to Up-@Level. Clicking on this button raises the operand level. Similarly, clicking the button labeled with a downward-pointing arrow lowers the operand level by calling Down-@Level. Both Up-@Level and Down-@Level consult the numeric prefix argument to determine the number of levels to move (the default is 1). If the option :wrap-@level is true, then the level will wrap around from top to bottom or bottom to top as necessary.

The current level is displayed on the panel. Depressing `Mouse_Right` over the displayed level causes a menu containing all of the levels to appear. The levels in the menu appear in order, with the highest level at the top of the menu.

To select a new level, continue to hold down `Mouse_Right`, move the mouse until the level that you desire is highlighted, and then release `Mouse_Right`. If no change is desired, move the mouse cursor away from the menu and release `Mouse_Right`.

The command Set-@Level-To-Character can be used to set the operand level to "Character".

3.10.2 Setting the Selection

Mouse-Select-@Level	Mouse_Left
Mouse-Extend-@Level	Mouse_Middle
Select-Region-Dot-To-Mark	Esc-^W
Select-Word	Esc-@
Select-Buffer	^X-h

Mouse-Select-Line	⟨Select,"Line"⟩
Mouse-Select-Word	⟨Select,"Word"⟩
Mouse-Select-Fullword	Esc-Mouse_Left
Mouse-Extend-Selection-Fullword	Esc-Mouse_Middle
Deselect-Region	Esc-^D

The current selection is set by the commands Mouse-Select-@Level, Mouse-Extend-@Level, and Select-Region-Dot-To-Mark. Mouse-Extend-@Level is normally bound to the middle button of the mouse. When the level is "Character", this command selects the region between the edit cursor and the mouse. At other levels, it selects the region that *includes* both the current selection and the operand (relative to the current operand level) beneath the mouse cursor. Select-Region-Dot-To-Mark selects the implicit region between the top mark on the mark stack and the edit cursor.

The command Select-Word sets the current selection to be the region from the edit cursor to the end of the word surrounding the edit cursor, while Mouse-Select-Fullword sets the current selection to be the full word beneath the mouse cursor. Mouse-Extend-Selection-Fullword extends the current selection to include the full word beneath the mouse cursor. Select-Buffer selects the entire buffer.

The current selection can be cleared using Deselect-Region.

3.10.3 Inserting Text

Self-Insert	most printable characters
Quote-Insert	^Q
Newline-And-Indent	^J
Insert-Newline	Return
Indent-Like-Previous-Line	Esc-Tab
Open-Line	^O
Split-Line	Esc-^O
Insert-Parentheses	(
Insert-Rparen-And-Match	)
Insert-Rbrace-And-Match	}
Insert-Rbracket-And-Match	]
Insert-File	^X-Tab
Insert-File	File: Insert from...

Typing a printable character generally causes that character to be inserted into the text at the position of the edit cursor. Quote-Insert inserts the next ASCII character typed.

Newline-And-Indent is bound to the newline character ^J; if the option :autoindent is true, the next line will be indented to the level of the previous line by inserting tabs and blanks. Indent-Like-Previous-Line simply reindents the current line to the level of the previous line. Open-Line inserts newlines after the cursor; the number of newlines inserted is determined by the value of the numeric prefix argument (default 1). Split-Line does the same thing, but also indents any text following the cursor to its original horizontal position.

The command Insert-Parentheses inserts a pair of matching parentheses at the cursor, and positions the cursor between them. The command Insert-Rparen-And-Match inserts a right parenthesis and briefly shows the matching left parenthesis by moving the cursor if the match is visible in the viewer or by displaying the line containing the match as a message on the panel otherwise. Insert-Rbrace-And-Match and Insert-Rbracket-And-Match perform the same action for braces and brackets.

Insert-File prompts for a file name and copies the contents that file into the active buffer at the position of the edit cursor.

3.10.4 Filling Text

Set-Auto-Fill-Column	^X-f
Toggle-Auto-Fill	^X-^A

Pan has a rudimentary mechanism for filling text lines as you type them. When auto-filling is on, the keys Space and Return are bound to special procedures. These procedures compare the current horizontal position of the cursor with the value of :auto-fill-column. If the line is too long, it will be broken where appropriate; if not, the procedures act like Self-Insert.

Use Toggle-Auto-Fill to turn on autofilling and again to turn it back off. When filling is on, you can type text continuously without worrying about line length. Use Set-Auto-Fill-Column (with a numeric prefix argument) to set the maximum line length used by auto-filling.

3.10.5 Deleting Text

Delete-@Level	^D
Delete-Character	⟨Delete,"Character"⟩
Delete-Previous-Character	Delete
Delete-Word	⟨Delete,"Word"⟩
Delete-Fullword	Not bound
Delete-Previous-Word	Not bound
Delete-Line	⟨Delete,"Line"⟩
Delete-Selected-Region	Edit: Delete
Delete-Region-Dot-To-Mark	Not bound
Delete-Blank-Lines	^X-^O
Delete-Horizontal-Space	Esc-\
Just-One-Space	Esc-Space
Delete-Indentation	Esc-^

Deleted text can be recovered by issuing the Undo command immediately after the deletions, but in no other way. Killed text is retained in the kill ring. The standard bindings reflect this limitation by using kill commands when removing large regions. *Pan* merges deletions that are contiguous in space and time into a single undoable action.

Delete-Character deletes the character selected by the active cursor; Delete-Previous-Character deletes the character before the cursor. The pair of commands Delete-Word and Delete-Previous-

Word delete from the cursor to the end (beginning) of the word enclosing the cursor, while Delete-Fullword deletes the entire word enclosing the cursor. The commands Delete-Selected-Region and Delete-Region-Dot-To-Mark operate on the selection and the implicit selection, respectively.

There are several ways to delete white space around the cursor. Delete-Blank-Lines deletes vertical and horizontal white space, leaving exactly one blank line at the cursor. Delete-Horizontal-Space deletes white space surrounding the cursor on the same line as the cursor; Just-One-Space does the same thing, but leaves exactly one space at the cursor. Delete-Indentation removes any leading white space on the line containing the cursor.

3.10.6 Killing Text

Kill-Word	`Esc-d`
Kill-Previous-Word	`Esc-Del`
Kill-To-EOL	`^K`
Kill-Selected-Region	`^W`
Kill-Selected-Region	Edit: `Kill`
Kill-Region-Dot-To-Mark	Not bound
Copy-Selection-As-Kill	`Esc-w`
Yank-From-Kill-Ring	`^Y`
Cycle-Yank	`Esc-Y`
Cycle-Kill	Not bound
Show-Kill	`^X-?`
Cycle-Show-Kill	`^X-!`

Like the deletion commands of the previous section, commands that kill text remove the text from an object. Unlike deletion commands, however, commands that kill text also place the removed text into the kill-ring.

The pair of commands Kill-Word and Kill-Previous-Word kill from the cursor to the end (beginning) of the word enclosing the cursor. The commands Kill-Selected-Region and Kill-Region-Dot-To-Mark kill the selection and the implicit selection, respectively. Kill-To-EOL kills all characters up to the end of the line.

Text in the kill ring can be recovered using the command Yank-From-Kill-Ring which inserts the contents of the top of the kill-ring into the buffer at the current cursor position. Cycle-Yank cycles the kill ring before yanking the top of thee kill-ring. Copy-Selection-As-Kill copies the current selection to the kill-ring.

The commands Show-Kill, Cycle-Kill, and Cycle-Show-Kill can be used to view the contents of the kill ring, and to cycle the ring.

3.10.7 The Clipboard

Cut-To-Clipboard	L10 (Delete)
Cut-To-Clipboard	Edit: Cut
Copy-To-Clipboard	L6 (Put)

Copy-To-Clipboard	Edit: Copy
Paste-From-Clipboard	L8 (Get)
Paste-From-Clipboard	Edit: Paste
Replace-From-Clipboard	Esc-L8
Cycle-Clipboard	Not bound
Show-Clipboard	Edit: Show Clipboard
Show-Clipboard	Esc-?
Cycle-Show-Clipboard	Esc-!

These commands manipulate the contents of the clipboard. Cut-To-Clipboard deletes the current selection and places it onto the clipboard. Copy-To-Clipboard places a copy of the current selection onto the clipboard. Paste-From-Clipboard inserts a copy of the clipboard's contents at the edit cursor. Finally, Replace-From-Clipboard replaces the selected region with the contents of the clipboard.

The command Show-Clipboard displays the contents of the clipboard in the help buffer. Cycle-Clipboard cycles the clipboard using the prefix argument to determine the number of entries to move, while Cycle-Show-Clipboard combines the two actions.

3.10.8 Copying and Moving Text

Copy-Selection-To-Cursor	Esc-^Y
Copy-Selection-To-Cursor	Edit: Copy To Cursor
Move-Selection-To-Cursor	Esc-^M
Move-Selection-To-Cursor	Edit: Move To Cursor

Text can copied or moved within a buffer by using either the clipboard or the local kill ring. However, the above commands are useful short cuts. Both use the current selection as the source to copy or move, and the edit cursor to mark the destination. These commands operate only within a buffer; copying and moving between buffers requires the clipboard.

3.10.9 Commands for Changing Case

Capitalize-Word	Esc-c
Lowercase-Word	Esc-l
Uppercase-Word	Esc-u
Capitalize-Selection	Esc-^C
Lowercase-Selection	Esc-^L
Uppercase-Selection	Esc-^U

These commands are use to change cases within a word or region. The word-oriented commands operate on the region from the cursor to the end of the word. They leave the cursor at the end of the region when done.

3.10.10 Transposing

Transpose-Characters	^T
Transpose-Previous-Characters	Not bound
Transpose-Lines	^X-^T

Transpose-Characters exchanges the character at the cursor and the character before the cursor. Transpose-Previous-Characters exchanges the two characters to the left of the cursor. Transpose-Lines exchanges the line containing the cursor with line before it.

3.11 Searching Text

Pan provides commands for searching for regular expressions and for matching balanced brackets.

3.11.1 Regular Expressions

Re-Search-Backward	^R
Re-Search-Forward	^S

These commands search for text matching the standard UNIX regular expressions. For a description of those expressions, see the ED(1) manual page of the UNIX Programmer's Manuals. *Pan* is unable to search for patterns which contain embedded newline characters. The most recently specified regular expression is shared by all buffers.

When text matching a pattern is found, the active cursor is moved to the first character in the match, and the matched text is selected.

Both Re-Search-Forward and Re-Search-Backward process the numeric prefix argument idiosyncratically: the *presence* of a prefix argument causes the command to search using the last regular expression specified. For instance, ^U-^S invokes Re-Search-Forward using the most recent search pattern—and the command will match the *next* occurrence of the pattern. Alternatively, supplying an empty string as the regular expression causes the previously specified expression to be used.

When the option :autowrap-search is true, searches wrap from one end of the buffer to the o' .er; if that option is false, searches terminate when finding the beginn.ng or end of the buffer.

While a Query-Replace command has not yet been implemented, one can rapidly perform that action by putting the replacement text into the clipboard, and then alternating searches with Replace-From-Clipboard.

3.11.2 Balanced Bracket Commands

Backward-Expr	Esc-^B
Forward-Expr	Esc-^F
Select-Expr	Esc-^@
Kill-Expr	Esc-^K
Show-Match	Esc-%

These commands use syntax class definitions to operate on balanced brackets. Both Forward-Expr and Backward-Expr move the cursor. The Select-Expr command selects the balanced bracket expression surrounding the cursor, and the command Kill-Expr kills it.

If the cursor character is a bracket, Show-Match moves the cursor to the matching bracket, pauses for the value of the option :pause-ticks internal ticks, and returns the cursor to its original position.

3.12 Editing Programs

Pan provides facilities for editing tree-structured objects described by a formal language. These objects include programs, which are described by a programming language. To take advantage of those facilities, the language being edited must be described to *Pan* using the lan⁻uage-definition language *Ladle*[4]. Language descriptions are beyond the scope of this manual. In ˙uis section, the basic language-oriented editing features of *Pan* are described.

Pan currently supports two languages: Modula-2[8] and ASPLE[5]. ASPLE is a simple example language used for demonstrations and for learning *Pan*. Other language descriptions, including *C*, are under development.

Full text editing is always available when editing structures. In fact, the actual operations that alter a structure are reduced to textual operations. Incremental syntactic analysis (parsing) then updates the internal tree structure. Most of the time, this transformation is hidden from the user—it occurs automatically as operations like "delete the selected subtree" are invoked. The next sections describe language-oriented editing with *Pan* in more detail.

3.12.1 Language-Oriented Viewers

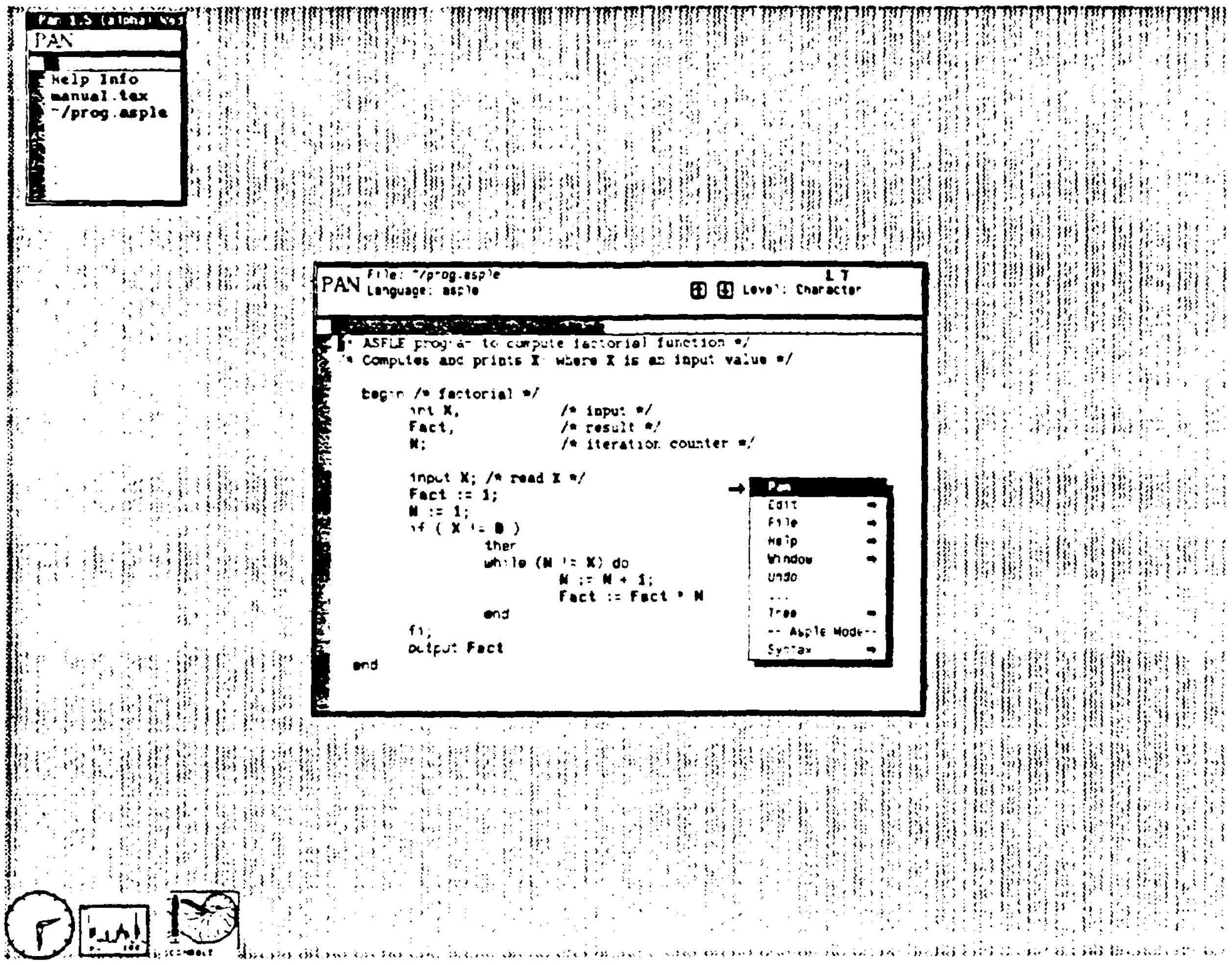
Structures are displayed using the same kind of viewer used to display text. A future version of *Pan* will provide a pretty-printing viewer that keeps the displayed structure consistent with its internal form. Right now, the re-indentation must be performed manually. Language-oriented viewers support the same scrolling operations as viewers onto textual objects.

Figure 4 shows a language-oriented viewer for a buffer containing an ASPLE program. Note that the "Language:" field in the information panel now reads "ASPLE", and that the root menu has been customized to contain the submenus Tree and Syntax. The Tree menu provides access to the basic tree navigation commands, while the Syntax menu provides language-oriented editing commands. As with text viewers, however, those commands are also available from the keyboard.

A language-oriented viewer has three new flags in the information panel. The "L" (:lex-ok) flag will be grey when there are changes in the buffer that have not yet been lexically analyzed (the first phase of parsing). The "T" (:tree-ok) flag will be visible, but grey, when there are changes that have not yet been incorporated into the tree by the parser. Both "L" and "T" flags are displayed in solid tones when the buffer's internal structures are consistent. The "!" (:parse-errors) flag appears when there are lexical or syntactic errors in the program being edited. Section 3.12.4 discusses this topic in more detail.

3.12.2 Selection

Selection during language-oriented editing relies on the notion of operand levels (section 2.3.3). In a language-oriented viewer, the set of operand levels is much richer than in the text world. The



File: ~/prog.asple
Language: asple
L T
Level: Character

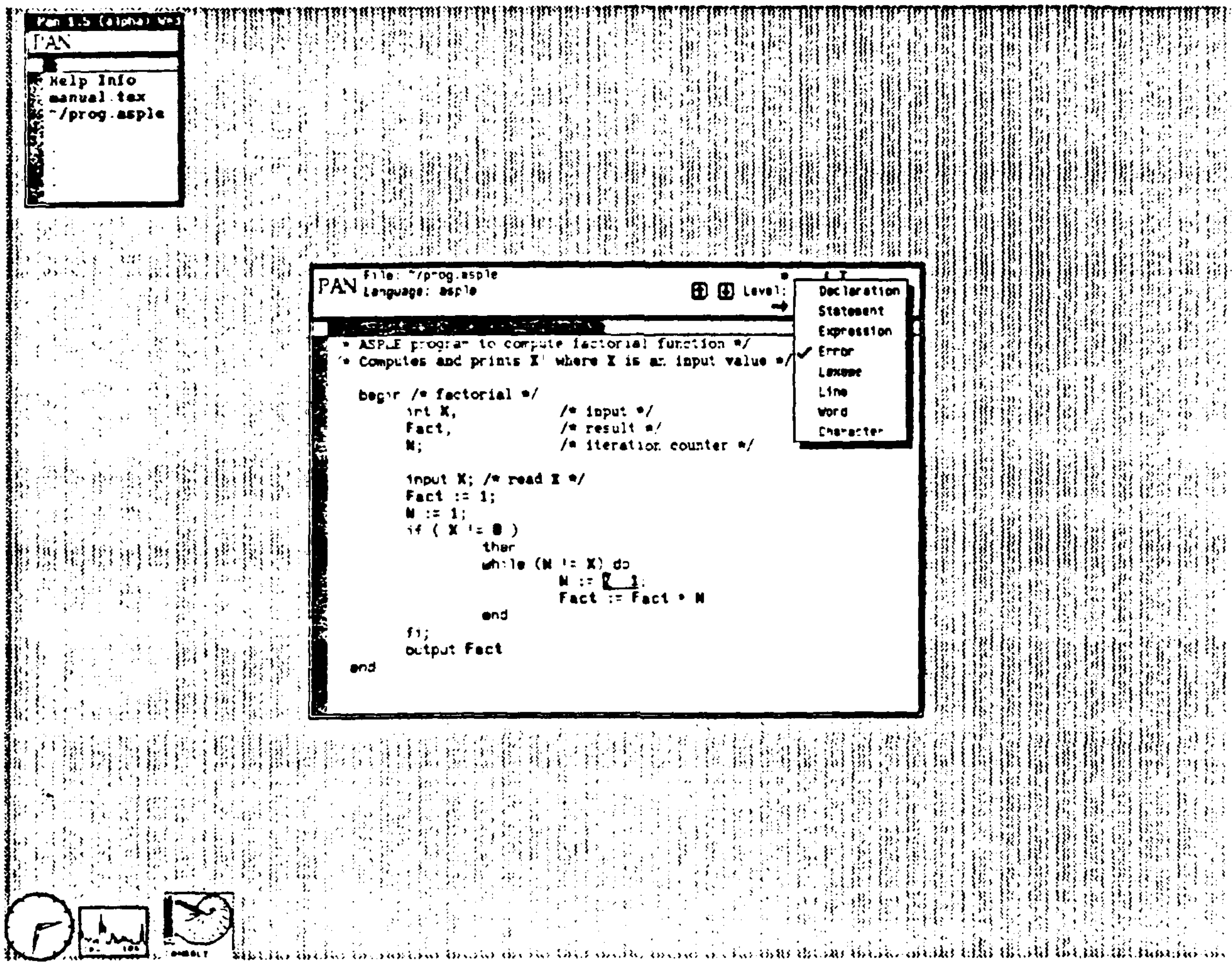
```
/* ASPLE program to compute factorial function */
/* Computes and prints X! where X is an input value */

  begin /* factorial */
        int X,              /* input */
        Fact,               /* result */
        N;                  /* iteration counter */

        input X; /* read X */
        Fact := 1;
        N := 1;
        if ( X != 0 )
                then
                while (N != X) do
                        N := N + 1;
                        Fact := Fact * N
                end
        fi;
        output Fact
 end
```

Figure 4: A viewer with a language other than "Text"

operand levels for text editing are included, as well as levels corresponding to basic abstractions in the language being edited. Figure 5 shows the set of operand levels defined for ASPLE. The "Error" level is of special significance: it is used to locate errors in a syntactic structure and to move the selection from one error to another.

As with text, depressing the left button of the mouse selects the object beneath the mouse cursor. The object actually chosen is determined by the setting of the operand level. Thus if the mouse cursor is over the character "*" in Figure 5 when the left button is clicked, the selected object might be the underlying lexeme, expression, or statement depending on whether the operand level is "Lexeme", "Expression", or "Statement" respectively.

What happens if the object beneath the mouse cursor is not an element of the class of objects specified by the current operand level? Then *Pan* uses a heuristic to find an object close to the mouse cursor that is an element of that class. This behavior is fairly predictable, although in some cases it leads to unforseen selections.



File: ~/prog.asple
Language: asple

```
* ASPLE program to compute factorial function */
/* Computes and prints X! where X is an input value */

 begin /* factorial */
       int X,               /* input */
       Fact,                /* result */
       N;                   /* iteration counter */

       input X; /* read X */
       Fact := 1;
       N := 1;
       if ( X != 0 )
               then
               while (N != X) do
                       N := N + 1;
                       Fact := Fact * N
               end
       fi;
       output Fact
end
```

Figure 5: The operand levels for ASPLE

To select arbitrary portions of a structure, set the operand level to "Character" and operate on the textual representation.

3.12.3 Navigation

Next-@Level	^F
Previous-@Level	^B
Tree-Up	R6
Tree-Up	Tree: Up
Tree-Down	R14
Tree-Down	Tree: Down
Tree-Left	R9
Tree-Left	Tree: Left

Tree-Right	R11
Tree-Right	Tree: Right

Pan provides two sets of language-oriented navigation operations. The most useful set involves the operations Next-@Level and Previous-@Level. They perform preorder and inverse-preorder tree walking operations relative to the current operand level. For example, if the operand level setting is "Error", the command Next-@Level moves the cursor to the "next" error node in the tree.

A second set of navigation commands includes Tree-Up, Tree-Down, Tree-Left, and Tree-Right. These are the usual tree-oriented commands. Tree-Up moves the cursor to the parent, Tree-Down moves it to the leftmost child of the current node, Tree-Left and Tree-Right move the cursor to the appropriate sibling in the tree. These commands can be used to explore the actual tree structure, as opposed to the tree structure imposed by the operand hierarchy mechanism. Use them carefully!

3.12.4 Parsing and Syntactic Errors

Parse-Buffer	Syntax: Parse Buffer
Rectify-Tree	Not bound

Parsing incorporates changes in a buffer's contents into the tree that represents a program. The parsing method used is incremental—only the areas affected by the changes are reparsed. Parsing occurs whenever a language-oriented operation takes place, such as when the operand level is changed to be a non-textual level or when a tree-oriented navigation command is invoked. Parsing can also be invoked manually, using the command Parse-Buffer. The command Rectify-Tree is used internally to invoke parsing if there have been any changes in the contents of the buffer.

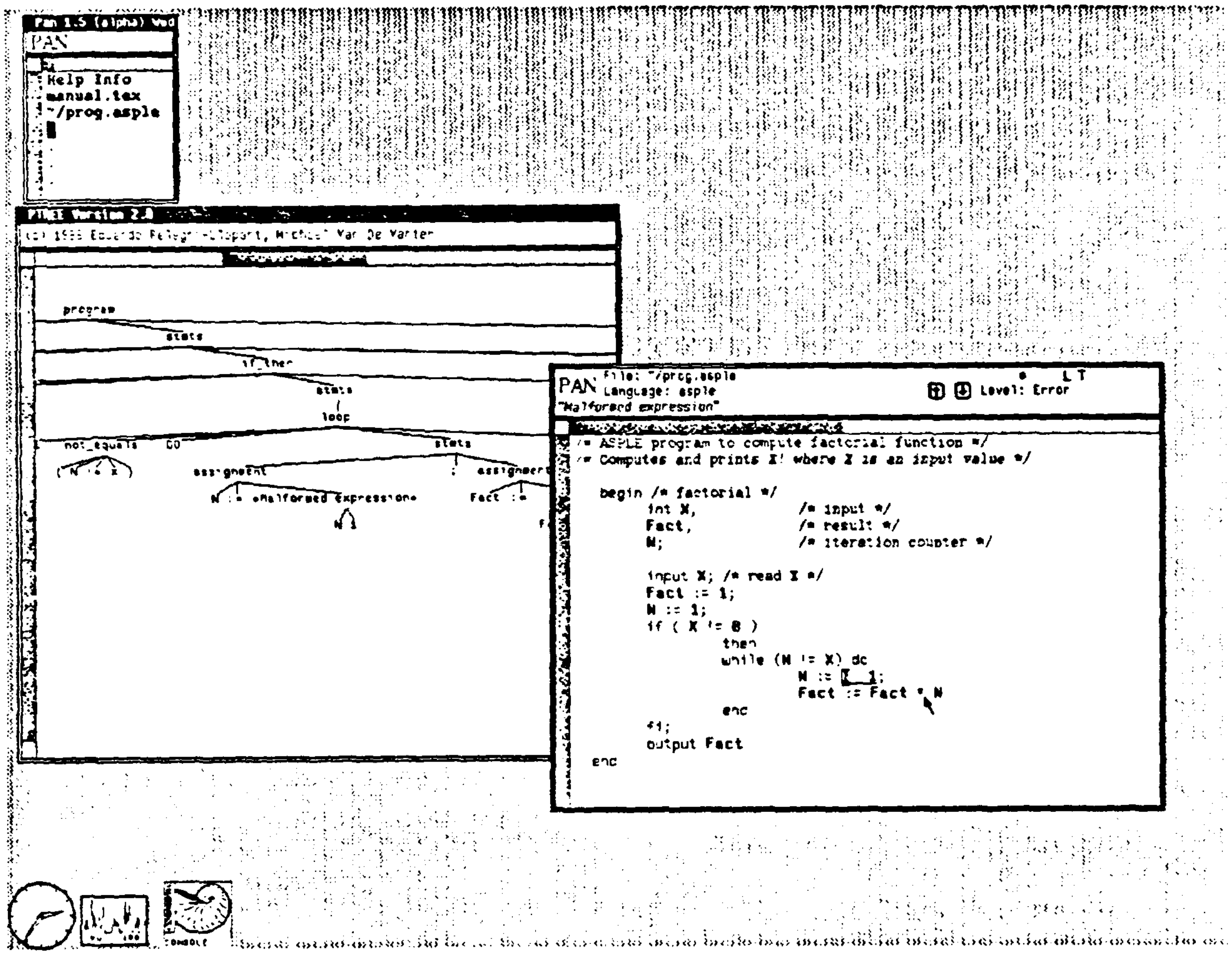
Parsing is a two-stage process. In the first stage, the text stream is broken into larger fragments called lexemes. Lexemes are the basic symbols in the language being edited, e.g., keywords, identifiers, constants, and comments. When a buffer is lexically analyzed, various classes of lexemes are given different visual images using fonts. The following table defines the relationship between font codes and characters in lexemes:

Class	Example	Font
Unanalyzed characters	inserted text	0
Ignored by the lexical analyzer		1
Fixed-length lexemes	keywords	2
Recognized but not parsed	comments	3
Variable-length lexemes	identifiers	4

Unanalyzed characters are always displayed using the default font 0.

After lexing, provided that there are no lexical errors, the parser updates the internal structured representation. If the parser encounters syntactic errors, the number of errors discovered during the parse is displayed on the annunciator line, and the :parse-errors flag is set. This flag appears as a "!" on the information panel.

When the parser detects an error, the subtrees involved in the error are gathered into an "error subtree". Selecting the subtree rooted at an error node causes the error message from the parser to be displayed on the annunciator line. Figure 6 shows the display of an error.



```
PAN File: ~/prog.asple                                   •    L T
    Language: asple                          Level: Error
"Malformed expression"

/* ASPLE program to compute factorial function */
/* Computes and prints X! where X is an input value */

  begin /* factorial */
        int X,                 /* input */
        Fact,                  /* result */
        N;                     /* iteration counter */

        input X; /* read X */
        Fact := 1;
        N := 1;
        if ( X != 0 )
                then
                while (N != X) do
                        N := X  1;
                        Fact := Fact * N
                end
        fi;
        output Fact
 end
```

Figure 6: Displaying an error in a program

As noted previously, the operand level can be set to "Error" in order to find and select errors in the tree. Figure 6 also shows a tool displaying the internal *tree structure* for the error. Generating such displays using *Ptree* is discussed in Section 3.12.6.

3.12.5 Editing

Delete-@Level ^D

Editing structured objects is quite simple in this version of *Pan*. All insertions use the text-level commands used for text editing. Top-down tree elaboration, being designed for the next major release, is not yet available. Deletions are accomplished using either text-level commands, or using the Delete-@Level command. Undoing edit actions restores the buffer without restoring the structured representation. Thus undoing currently requires reparsing.

3.12.6 Displaying Trees using *Ptree*

Print-Parse-Tree	Syntax: `Print Parse Tree`

Ptree is a separate program that displays tree structures. A *Ptree* window appears in Figure 6. *Ptree* is not an integral part of *Pan* at present, although a *Ptree*-like viewer for *Pan* is planned. *Ptree* runs as a separate tool in the SunView environment. The manual page PTREE(1) provides more details on its operation. *Ptree* displays are primarily a debugging tool for the authors of *Ladle* language descriptions.

Pan can be used to create *Ptree* input. The command Print-Parse-Tree formats the internal tree structure of the active buffer into a file called `ptree.out` in the working directory. This file can then be displayed using *Ptree*.

3.13 When Things Go Wrong

Oops	Not bound
Rats!	Not bound

Pan has been remarkably (well, reasonably) robust throughout its long development period. Most problems are routinely handled by printing a message on the message line of the active viewer. However, during early-release, provisions have been made for recovering from major catastrophes.

We'd probably all agree that a catastrophe has occurred if *Pan* failed either by returning to the underlying *Lisp* system or by dying altogether. Fortunately, the first rarely happens, and the second won't occur without returning to *Lisp*.

If the option :break-to-lisp is true and an unanticipated error occurs, *Pan* enters a *Lisp* break loop. When *Pan* encounters an internal error and the option :save-on-system-error is true, *Pan* attempts to save all of the modified buffers. These two options are currently configured so that a break loop is entered, and saving is turned off. The break loop is entered before any saving of files is attempted.

If you do somehow end up in a *Lisp* break loop, a prompt will appear in the tool window in which the system is running. (If *Pan* is running from a menu, the prompt will appear in the console tool.). The prompt will look like "{*nn*}" or "*nn* ⇒" where "*nn*" is a small integer. In the first case ("{*nn*}"), the system is in a *Lisp* break loop. You can recover from the error by typing the *Lisp* expression `(Oops)`. This returns the system to the normal command evaluation loop. Naturally, the circumstances of such an error should be noted and passed on to the developers of *Pan*. (Mail to panpipes@renoir.)

The second case ("*nn* ⇒") is more serious. In fact, the session is almost over. All that you can do is to type `(Rats!)`. This command executes the normal code for saving the modified buffers and then exits the system.

4 Simple Customization

Pan can be customized by altering option values and bindings, extended by defining new options, flags, and commands, and broadened by defining new, formal languages by using the language definition language *Ladle*. This section provides a brief introduction to the facilities for tailoring and extending the system.

4.1 Start-Up Processing

Auto-Load *file-name regular-expression*
Auto-Exec *function-name regular-expression*
Load-File ^X-^L

At start-up, a run-command file named `.panrc` is loaded into *Pan*. The `.panrc` file should be a file of *Lisp* and *Pan* commands located either in your working or your home directory.

The `.panrc` file is loaded using the command Load-File. All commands that load files use the search path specified by the option :pan-load-search-path. The default value of the option :pan-load-search-path is set to `(. ~ ~piper/lib/pan )`.

Pan can be instructed to automatically load files other than `.panrc`. One way is to include Load-File directives in the `.panrc` file. Such files will be loaded once at start-up. This method can be used to ensure that a certain selection of libraries will always be loaded. A second way is to use the Auto-Load command.

The Auto-Load command instructs *Pan* to ensure that a file has been loaded whenever a buffer whose name matches a given UNIX file expression is created. The file is loaded at most once as a result of Auto-Load.

For instance, `(Auto-Load "c-lib" "*.[hc]")` tells the system to load the file "`c-lib`" the first time that a file whose name matches "`*.[hc]`" is edited. The file `c-lib` can be either lisp code or compiled lisp code; if both `c-lib.l` and a `c-lib.o` are found in the same directory, the most recently modified version is chosen.

Similar to Auto-Load, the command Auto-Execute instructs *Pan* to execute a given bindable *function* whenever a buffer having a name that matches a given pattern is created. Taken together, Auto-Load and Auto-Execute can be used to create minor modes.

A minor mode is a collection of commands and bindings useful while editing objects of a given type. For instance

```
(Auto-Load "tex-mode" "*.tex")
(Auto-Execute 'tex-mode "*.tex")
```

can be used to create a minor mode for TEX input. The parameterless function `tex-mode` is defined in the library file `tex-mode.l` to set up a specialized collection of bindings. The file `tex-mode.l` also defines a number of commands useful for manipulating TEXnical text.

4.2 Bindings

Bind-To-Key *command key-sequence [location]*
Bind-To-Menu *command menu-name [label location]*
Add-Menu-To-Menu *menu-being-added to-menu [location]*

Define-Operand-Levels *levels*
Add-Operand-Levels *levels*

Bindings are established or altered using the functions Bind-To-Key, Bind-To-Menu, and Add-Menu-To-Menu. The default location of the change is in the local buffer; specifying `:global` makes the change in the global environment. The commands Add-Menu-To-Menu, Define-Operand-Levels, and Add-Operand-Levels are described in *The Pan Extension Manual.*

The syntax for Bind-To-Key is

(`Bind-To-Key` *'function "key sequence" [location]*)

where *'function* is a quoted function name, *"key sequence"* is a *Lisp* string specifying a keystroke sequence, and *location* is either `:local` or `:global`. The default value for *location* is `:local`.

When specifying a keystroke sequence, control characters such as "`Control-?`" are denoted by the two-character sequence "`^?`"; the `Escape` prefix is denoted by `Esc-`. In multiple keystroke sequences, the keystrokes must be separated by hyphens. For example,

```
(Bind-To-Key 'Delete-Character "^D" :global)
```

establishes the default binding for Delete-Character.

To establish a binding to a menu item, use

(`Bind-To-Menu` *'function "Menu Name" ["Selection Name" location]*)

Both parts of the menu item must be *Lisp* strings. If *"Selection Name"* is omitted, the name of the command is used. The argument *location* is identical to the argument of the same name for the Bind-To-Key command. For example,

```
(Bind-To-Menu 'Describe-Selection "Help" "Describe" :global)
```

sets up the default binding for Describe-Selection in the Help menu.

Show-Key-Bindings	Help: `Key Bindings`
Show-Menu-Bindings	Help: `Menu Bindings`
Show-Operand-Bindings	Help: `Operand Bindings`

Show-Key-Bindings, Show-Menu-Bindings, and Show-Operand-Bindings print the appropriate bindings visible in the current buffer into the help buffer.

It is possible to invoke a command without binding it by using the commands Execute-Lisp-Line or Execute-Named-Command. (The next section provides more details.)

4.3 Getting and Setting Option Values

Option-Value *option-name [lookup buffer]*

The command Option-Value is used to retrieve (and set) option values. Its syntax is

(`Option-Value` *option-name [lookup buffer]*) where *option-name* is the name of the option, and *lookup* and *buffer* are optional arguments. The argument *lookup*, if specified, must be one of `:local`, `:global`, or `:default`. When *lookup* is left unspecified, the value `:default` (look first for a definition in the active buffer; if none is found, look for a global definition) is used. *Buffer* defaults to the active buffer.

To set an option's value to *value*, use the lisp form

(setf (Option-Value *option-name [lookup buffer]*) *value*)

In this case, *lookup* must specify either `:local` or `:global`; it defaults to `:local`. For instance, the expression

```
(setf (Option-Value :minor-mode :local) "TeX Mode")
```

sets the value of the option :minor-mode in the active buffer to be `"TeX Mode"`.

4.4 Lisp-Oriented Commands

These commands are for the use of people extending *Pan*, although sometimes you'll want to use one to see what a command does. They are included here for completeness.

Execute-Lisp-Line	Esc-Esc
Execute-Named-Command	Esc-x
Load-File	^X-^L

Execute-Lisp-Line prompts for a *Lisp* expression to evaluate; the result is printed on the message line. If you want to execute a bindable *Pan* command, there are two methods. The simplest is to invoke Execute-Named-Command and respond to the prompt with the name of the command, e.g. `Next-Character`. Alternatively, one can invoke Execute-Lisp-Line and respond to the prompt with the expression `(Next-Character)`. The parentheses are required in the latter case.

Load-File loads a file of lisp and *Pan* commands into the system.

5 Acknowledgments

Many have helped with the creation of *Pan*. Thanks especially to Jacob Butcher and Christina Black. Jacob implemented the language-description processor *Ladle* and the tree data structures. Christina is developing the pretty-printing viewer and helped with the preparation of these reports. Eduardo Pelegrí-Llopart and Phillip Garrison have also made many valuable suggestions.

References

[1] *Windows and Window Based Tools: Beginner's Guide.* Sun Microsystems, Inc., 1986.

[2] Robert A. Ballance and Michael L. Van De Vanter. The Pan extension manual. In preparation.

[3] Robert A. Ballance, Michael L. Van De Vanter, and Susan L. Graham. *The Architecture of Pan I.* Technical Report 88/409, Computer Science Division, UC Berkeley, March 1988.

[4] Jacob Butcher. Ladle. In preparation.

[5] J. C. Cleaveland and R. C. Uzgalis. *Grammars for Programming Languages.* Elsevier Holland, 1977.

[6] R. M. Stallman. EMACS, the extensible, customizable, self-documenting display editor. In *Proc. of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, pages 147–156, 1981.

[7] Richard Stallman. *GNU Emacs Manual: Fifth Edition, Emacs Version 18 for Unix Users.* October 1986.

[8] Nicklaus Wirth. *Programming in Modula-2.* Springer-Verlag, third, corrected edition, 1985.

6 Glossary

Apropos A Help command for information gathered during command, flag, and option definition. For instance, `(Apropos 'Cursor)` lists all of the commands dealing with the cursor.

Base Buffer The base buffer is a special buffer that is the root for all *Pan* buffers. The base buffer has is the only frame that can be made iconic. Quitting the base buffer terminates *Pan*. All currently editable objects are listed in the base buffer.

Buffer A buffer is the locus of editing attention for a single editable object (currently a text file). Buffers contain a copy of the object being edited, key and menu bindings, a selection, viewers, and other objects.

Clipboard The clipboard is an area, shared by all buffers, that contains a region of text. It is used to implement cut and paste between buffers, or between *Pan* and other processes.

Command A command is a user-level procedure for effecting edit operations. Commands are defined using Define-Command.

Dot The dot is another name for the edit cursor. In effect, it is the integer offset (in characters) of the edit cursor from the beginning of the file.

Edit Cursor A cursor is an object marking the location where alterations of the edit object can occur. In particular, characters are inserted or deleted at the character position to the left of the character designated by the cursor.

Edit Window The area of a viewer in which the object being edited is displayed.

Flag A flag is a user-definable object that stores a single bit of information. Flags can be defined having a user-visible representation on the information panel.

Frame The outer surrounding edge of a viewer that responds to SunView window protocols.

Help Buffer A special buffer used for displaying help information.

Mark A mark is a character position in a text file. Marks are generally used to remember positions for later processing. *Pan* provides for a stack of marks. The top mark in the stack is known as "the mark".

Operand Level The operand level of a viewer designates the type of operand to be used by generic operations. For instance, if the operand level is "Word", then the Next-@Level command moves the cursor to the next word in the object being edited.

Option An option is a user-definable typed variable. Many of the customizations available to a user are provide via predefined options. Unlike a flag, an option does not have a visible presentation. However, options can have special "notifier" functions that are called whenever the option value is changed.

Region A region is a contiguous sequence of characters. Most text operations involve regions either as source, destination, or both.

Selection A specially designated region. There is one selection per buffer.

Viewer The counterpart to a window in *Emacs*, a viewer displays an object. Each viewer has its own edit cursor and display state.

A Default Key Bindings

A.1 Bindings By Command Name

Abort-Command	Esc-^G
Abort-Command	^C-^G
Abort-Command	^G
Abort-Command	^X-^G
Abort-Command	^Z-^G
Backward-Expr	Esc-^B
Backward-Vscroll	Esc-v
Capitalize-Selection	Esc-^C
Capitalize-Word	Esc-c
Close-Active-Viewer	^X-0
Copy-Selection-As-Kill	Esc-w
Copy-Selection-To-Cursor	Esc-^Y
Copy-To-Clipboard	L6
Cut-To-Clipboard	L10
Cycle-Show-Clipboa d	Esc-!
Cycle-Show-Kill	^X-!
Cycle-Yank	Esc-y
Delete-@Level	^D
Delete-Blank-Lines	^X-^O
Delete-Horizontal-Space	Esc-\
Delete-Indentation	Esc-^
Delete-Previous-Character	Backspace
Delete-Previous-Character	Del
Deselect-Region	Esc-^D
Down-@Level	F3
Execute-From-Menu	Mouse_Right
Execute-Lisp-Line	Esc-Esc
Execute-Named-Command	Esc-x
First-Non-Blank	Esc-m
Forward-Expr	Esc-^F
Forward-Vscroll	^V
Goto-Line	^X-1
Indent-Like-Previous-Line	Esc-Tab
Insert-File	^X-Tab
Insert-Newline	Return
Insert-Parentheses	Esc-(
Insert-Rbrace-And-Match	}
Insert-Rbracket-And-Match	]
Insert-Rparen-And-Match	)
Interrupt-Pan	^X-^Z
Just-One-Space	Esc-Space
Kill-Expr	Esc-^K
Kill-Previous-Word	Esc-Del
Kill-Selected-Region	^W
Kill-To-EOL	^K
Kill-Word	Esc-d
Left-Hscroll	^X-<
List-Files	^X-^D
Load-File	^X-^L
Lowercase-Selection	Esc-^L
Lowercase-Word	Esc-l
Mouse-Extend-@Level	Mouse_Middle
Mouse-Select-@Level	Mouse_Left
Mouse-Extend-Selection-Fullword	Esc-Mouse_Middle
Mouse-Select-Fullword	Esc-Mouse_Left
Move-Selection-To-Cursor	Esc-Return
Move-To-BOB	Esc-<
Move-To-BOL	^A
Move-To-EOB	Esc->
Move-To-EOL	^E
Newline-And-Indent	Newline
Next-@Level	^F
Next-Line	^N
Next-Word	Esc-f
Open-Another-Viewer	^X-2
Open-Line	^O
Paste-From-Clipboard	L8
Previous-@Level	^B
Previous-Line	^P
Previous-Word	Esc-b
Quote-Insert	^Q
Re-Search-Backward	^R
Re-Search-Forward	^S
Read-Prefix-Arguments	^U
Redraw	^L
Remove-Buffer	^X-k
Replace-From-Clipboard	Esc-L8
Right-Hscroll	^X->
Save-All-Buffers	^X-Return
Save-Buffer-File	^X-^S
Select-Buffer	^X-h
Select-Expr	Esc-^@
Select-Region-Dot-To-Mark	Esc-^W
Select-Word	Esc-@
Self-Insert	*-\
Self-Insert	Space-(
Self-Insert	Tab
Self-Insert	^--|
Self-Insert	-
Set-@Level-To-Character	F1
Set-Auto-Fill-Column	^X-f
Set-Mark	^@
Show-Buffers	^X-^B
Show-Clipboard	Esc-?
Show-Kill	^X-?

Show-Match	Esc-%
Show-Status-Line	^X-=
Split-Line	Esc-^O
Sun-Again	L2
Sun-Expose	L5
Sun-Find	L9
Sun-Open	L7
Sun-Props	L3
Sun-Stop	L1
Swap-Dot-And-Mark	^X-^X
Toggle-Auto-Fill	^X-^A
Toggle-Read-Only	^X-^Q
Transpose-Characters	^T
Transpose-Lines	^X-^T
Undo	L4
Undo	^X-u
Up-@Level	F2
Uppercase-Selection	Esc-^U
Uppercase-Word	Esc-u
Visit-Buffer	^X-b
Visit-File	^X-^F
Write-Files-Exit	^X-^C
Write-To-File	^X-^W
Yank-From-Kill-Ring	^Y

A.2 Bindings By Key

^@	Set-Mark
^A	Move-To-BOL
^B	Previous-@Level
^C-^G	Abort-Command
^D	Delete-@Level
^E	Move-To-EOL
^F	Next-@Level
^G	Abort-Command
^K	Kill-To-EOL
^L	Redraw
^N	Next-Line
^O	Open-Line
^P	Previous-Line
^Q	Quote-Insert
^R	Re-Search-Backward
^S	Re-Search-Forward
^T	Transpose-Characters
^U	Read-Prefix-Arguments
^V	Forward-Vscroll
^W	Kill-Selected-Region
^X-!	Cycle-Show-Kill
^X-0	Close-Active-Viewer
^X-2	Open-Another-Viewer
^X-<	Left-Hscroll
^X-=	Show-Status-Line
^X->	Right-Hscroll
^X-?	Show-Kill
^X-Return	Save-All-Buffers
^X-Tab	Insert-File
^X-^A	Toggle-Auto-Fill
^X-^B	Show-Buffers
^X-^C	Write-Files-Exit
^X-^D	List-Files
^X-^F	Visit-File
^X-^G	Abort-Command
^X-^L	Load-File
^X-^O	Delete-Blank-Lines
^X-^Q	Toggle-Read-Only
^X-^S	Save-Buffer-File
^X-^T	Transpose-Lines
^X-^W	Write-To-File
^X-^X	Swap-Dot-And-Mark
^X-^Z	Interrupt-Pan
^X-b	Visit-Buffer
^X-f	Set-Auto-Fill-Column
^X-h	Select-Buffer
^X-k	Remove-Buffer
^X-l	Goto-Line
^X-u	Undo
^Y	Yank-From-Kill-Ring
^Z-^G	Abort-Command
^_|	Self-Insert
}	Insert-Rbrace-And-Match
)	Insert-Rparen-And-Match
]	Insert-Rbracket-And-Match
-	Self-Insert
*-\	Self-Insert
Backspace	Delete-Previous-Character
Del	Delete-Previous-Character
Esc-!	Cycle-Show-Clipboard
Esc-(	Insert-Parentheses
Esc-<	Move-To-BOB
Esc->	Move-To-EOB
Esc-?	Show-Clipboard
Esc-@	Select-Word
Esc-Del	Kill-Previous-Word
Esc-Esc	Execute-Lisp-Line
Esc-L8	Replace-From-Clipboard
Esc-Mouse_Left	Mouse-Select-Fullword
Esc-Mouse_Middle	Mouse-Extend-Selection-Fullword
Esc-Return	Move-Selection-To-Cursor
Esc-Space	Just-One-Space
Esc-Tab	Indent-Like-Previous-Line
Esc-%	Show-Match
Esc-\	Delete-Horizontal-Space
Esc-^@	Select-Expr
Esc-^B	Backward-Expr
Esc-^C	Capitalize-Selection
Esc-^D	Deselect-Region
Esc-^F	Forward-Expr
Esc-^G	Abort-Command
Esc-^K	Kill-Expr
Esc-^L	Lowercase-Selection
Esc-^O	Split-Line
Esc-^U	Uppercase-Selection
Esc-^W	Select-Region-Dot-To-Mark
Esc-^Y	Copy-Selection-To-Cursor
Esc-^	Delete-Indentation
Esc-b	Previous-Word
Esc-c	Capitalize-Word
Esc-d	Kill-Word
Esc-f	Next-Word
Esc-l	Lowercase-Word
Esc-m	First-Non-Blank
Esc-u	Uppercase-Word
Esc-v	Backward-Vscroll
Esc-w	Copy-Selection-As-Kill

`Esc-x`	Execute-Named-Command
`Esc-y`	Cycle-Yank
`F1`	Set-@Level-To-Character
`F2`	Up-@Level
`F3`	Down-@Level
`L10`	Cut-To-Clipboard
`L1`	Sun-Stop
`L2`	Sun-Again
`L3`	Sun-Props
`L4`	Undo
`L5`	Sun-Expose
`L6`	Copy-To-Clipboard
`L7`	Sun-Open
`L8`	Paste-From-Clipboard
`L9`	Sun-Find
`Mouse_Left`	Mouse-Select-@Level
`Mouse_Middle`	Mouse-Extend-@Level
`Mouse_Right`	Execute-From-Menu
`Newline`	Newline-And-Indent
`Return`	Insert-Newline
`Space-(`	Self-Insert
`Tab`	Self-Insert

B Default Menu Bindings in a Text Buffer

B.1 Bindings by Command Name

Append-Selection-To-File	File: Append Selection To ...
Apropos-All-Symbols	Help: Apropos All
Apropos-Selection	Help: Apropos
Close-Active-Viewer	Window: Close
Copy-Selection-To-Cursor	Edit: Copy To Cursor
Copy-To-Clipboard	Edit: Copy
Cut-To-Clipboard	Edit: Cut
Delete-Selected-Region	Edit: Delete
Describe-Operand-Hierarchy	Help: Operand Hierarchy
Describe-Selection	Help: Describe
Frame-Edit-Cursor	Window: Frame Edit Cursor
Insert-File	File: Insert from...
Kill-Selected-Region	Edit: Kill
List-All-Commands	Help: Commands
List-Files	File: Directory list
Move-Selection-To-Cursor	Edit: Move To Cursor
Open-Another-Viewer	Window: Open Another Viewer
Overwrite-File!	File: Overwrite ...
Paste-From-Clipboard	Edit: Paste
Reset-Help-Buffer	Help: Reset Help Buffer
Save-Buffer-File	File: Save
Show-Buffers	Help: Buffers
Show-Clipboard	Edit: Show Clipboard
Show-Flag-Values	Help: Flag Values
Show-Key-Bindings	Help: Key Bindings
Show-Menu-Bindings	Help: Menu Bindings
Show-Operand-Bindings	Help: Operand Bindings
Show-Option-Values	Help: Option Values
Undo	Pan: Undo
Visit-File	File: Visit File
Write-Selection-To-File	File: Write Selection To ...
Write-To-File	File: Write To ...

B.2 Bindings by Menu

Edit: Copy To Cursor	Copy-Selection-To-Cursor
Edit: Copy	Copy-To-Clipboard
Edit: Cut	Cut-To-Clipboard
Edit: Delete	Delete-Selected-Region
Edit: Kill	Kill-Selected-Region
Edit: Move To Cursor	Move-Selection-To-Cursor
Edit: Paste	Paste-From-Clipboard
Edit: Show Clipboard	Show-Clipboard
File: Append Selection To ...	Append-Selection-To-File
File: Directory list	List-Files

File: `Insert from...`	Insert-File
File: `Overwrite ...`	Overwrite-File!
File: `Save`	Save-Buffer-File
File: `Visit File`	Visit-File
File: `Write Selection To`	Write-Selection-To-File
File: `Write To ...`	Write-To-File
Help: `Apropos All`	Apropos-All-Symbols
Help: `Apropos`	Apropos-Selection
Help: `Buffers`	Show-Buffers
Help: `Commands`	List-All-Commands
Help: `Describe`	Describe-Selection
Help: `Flag Values`	Show-Flag-Values
Help: `Key Bindings`	Show-Key-Bindings
Help: `Menu Bindings`	Show-Menu-Bindings
Help: `Operand Bindings`	Show-Operand-Bindings
Help: `Operand Hierarchy`	Describe-Operand-Hierarchy
Help: `Option Values`	Show-Option-Values
Help: `Reset Help Buffer`	Reset-Help-Buffer
Pan: `Undo`	Undo
Window: `Close`	Close-Active-Viewer
Window: `Frame Edit Cursor`	Frame-Edit-Cursor
Window: `Open Another Viewer`	Open-Another-Viewer

C Default Menu Bindings in the Base Buffer

C.1 Bindings by Command Name

Apropos-All-Symbols	Help: `Apropos All`
Apropos-Selection	Help: `Apropos`
Copy-To-Clipboard	Base Buffer: `Copy`
Describe-Operand-Hierarchy	Help: `Operand Hierarchy`
Describe-Selection	Help: `Describe`
List-All-Commands	Help: `Commands`
List-Files	Base Buffer: `Directory list`
Remove-Selected-Buffer	Base Buffer: `Remove-Selected-Buffer`
Reset-Help-Buffer	Help: `Reset Help Buffer`
Show-Buffers	Help: `Buffers`
Show-Flag-Values	Help: `Flag Values`
Show-Key-Bindings	Help: `Key Bindings`
Show-Menu-Bindings	Help: `Menu Bindings`
Show-Operand-Bindings	Help: `Operand Bindings`
Show-Option-Values	Help: `Option Values`
Undo	Base Buffer: `Undo`
Visit-File	Base Buffer: `Visit File`
Visit-Selection	Base Buffer: `Visit`

C.2 Bindings by Menu

Base Buffer: `Copy`	Copy-To-Clipboard
Base Buffer: `Directory list`	List-Files
Base Buffer: `Remove-Selected-Buffer`	Remove-Selected-Buffer
Base Buffer: `Undo`	Undo
Base Buffer: `Visit File`	Visit-File
Base Buffer: `Visit`	Visit-Selection
Help: `Apropos All`	Apropos-All-Symbols
Help: `Apropos`	Apropos-Selection
Help: `Buffers`	Show-Buffers
Help: `Commands`	List-All-Commands
Help: `Describe`	Describe-Selection
Help: `Flag Values`	Show-Flag-Values
Help: `Key Bindings`	Show-Key-Bindings
Help: `Menu Bindings`	Show-Menu-Bindings
Help: `Operand Bindings`	Show-Operand-Bindings
Help: `Operand Hierarchy`	Describe-Operand-Hierarchy
Help: `Option Values`	Show-Option-Values
Help: `Reset Help Buffer`	Reset-Help-Buffer

D Default Menu Bindings in the Help Buffer

D.1 Bindings by Command Name

Append-Selection-To-File	File: `Append Selection To ...`
Apropos-All-Symbols	Help: `Apropos All`
Apropos-Selection	Help Buffer: `Apropos`
Apropos-Selection	Help: `Apropos`
Close-Active-Viewer	Window: `Close`
Copy-To-Clipboard	Help Buffer: `Copy`
Describe-Operand-Hierarchy	Help: `Operand Hierarchy`
Describe-Selection	Help Buffer: `Describe`
Describe-Selection	Help: `Describe`
Frame-Edit-Cursor	Window: `Frame Edit Cursor`
List-All-Commands	Help: `Commands`
List-Files	File: `Directory list`
Open-Another-Viewer	Window: `Open Another Viewer`
Overwrite-File!	File: `Overwrite ...`
Reset-Help-Buffer	Help Buffer: `Reset Help Buffer`
Reset-Help-Buffer	Help: `Reset Help Buffer`
Show-Buffers	Help: `Buffers`
Show-Flag-Values	Help: `Flag Values`
Show-Key-Bindings	Help: `Key Bindings`
Show-Menu-Bindings	Help: `Menu Bindings`
Show-Operand-Bindings	Help: `Operand Bindings`
Show-Option-Values	Help: `Option Values`
Undo	Help Buffer: `Undo`
Visit-File	File: `Visit File`

Visit-Selection	File: `Visit`
Write-Selection-To-File	File: `Write Selection To ...`
Write-To-File	File: `Write To ...`

D.2 Bindings by Menu

File: `Append Selection To ...`	Append-Selection-To-File
File: `Directory list`	List-Files
File: `Overwrite ...`	Overwrite-File!
File: `Visit File`	Visit-File
File: `Visit`	Visit-Selection
File: `Write Selection To ...`	Write-Selection-To-File
File: `Write To ...`	Write-To-File
Help Buffer: `Apropos`	Apropos-Selection
Help Buffer: `Copy`	Copy-To-Clipboard
Help Buffer: `Describe`	Describe-Selection
Help Buffer: `Reset Help Buffer`	Reset-Help-Buffer
Help Buffer: `Undo`	Undo
Help: `Apropos All`	Apropos-All-Symbols
Help: `Apropos`	Apropos-Selection
Help: `Buffers`	Show-Buffers
Help: `Commands`	List-All-Commands
Help: `Describe`	Describe-Selection
Help: `Flag Values`	Show-Flag-Values
Help: `Key Bindings`	Show-Key-Bindings
Help: `Menu Bindings`	Show-Menu-Bindings
Help: `Operand Bindings`	Show-Operand-Bindings
Help: `Operand Hierarchy`	Describe-Operand-Hierarchy
Help: `Option Values`	Show-Option-Values
Help: `Reset Help Buffer`	Reset-Help-Buffer
Window: `Close`	Close-Active-Viewer
Window: `Frame Edit Cursor`	Frame-Edit-Cursor
Window: `Open Another Viewer`	Open-Another-Viewer

E Options Defined in *Pan*

:auto-fill OPTION
Type: `boolean`
Default Value: `nil`
T iff Space & Newline should cause long lines to be broken.

:auto-fill-column OPTION
Type: `fixp`
Default Value: `65`
Max line length for auto-fill.

:autoindent OPTION
Type: `boolean`

Default Value: `t`
T iff Newline-And-Indent should indent.

:autowrap-search OPTION
Type: `boolean`
Default Value: `t`
Wrap search from one end of file to other?

:backup-on-read OPTION
Type: `boolean`
Default Value: `t`
Make backup file when beginning to edit a file

:base-frame-font-map OPTION
Type: `listp`
Default Value: `("cour.b.14")`
Font map for base window.

:base-win-cols OPTION
Type: `fixp`
Default Value: 60
Default number of columns in base window.

:base-win-rows OPTION
Type: `fixp`
Default Value: 6
Default number of rows in base window.

:break-to-lisp OPTION
Type: `boolean`
Default Value: `t`
Break to underlying LISP on tragic error?.

:camel-debug OPTION
Type: `boolean`
Default Value: `nil`
Turn on internal parser tracing.

:checkpoint-freq OPTION
Type: `fixp`
Default Value: 450
Maximum number of command invocations before checkpointing.

:checkpoint-min OPTION
Type: `fixp`
Default Value: 400
Minimum number of command invocations prior to checkpointing.

:clipboard-max-size OPTION
Type: `fixp`
Default Value: 8
Maximum number of clips in clipboard

:col-popover OPTION
Type: `fixp`
Default Value: `16`
Minumum characters to move window when scrolling left/right.

:dont-reuse-nodes OPTION
Type: `boolean`
Default Value: `t`
Turn off node reuse during incremental parsing

:empty-line-char OPTION
Type: `fixp`
Default Value: `46`
Character prefix for empty lines.

:full-parse-tree OPTION
Type: `boolean`
Default Value: `nil`
Create full parse tree during parsing?

:help-frame-font-map OPTION
Type: `listp`
Default Value: `("cour.r.12")`
Font map for help windows.

:help-title OPTION
Type: `stringp`
Default Value: `"Help window"`
Title for help window

:help-win-cols OPTION
Type: `fixp`
Default Value: `60`
Default number of columns in-help window.

:help-win-rows OPTION
Type: `fixp`
Default Value: `20`
Default number of rows in help window.

:indent-with-tabs OPTION
Type: `boolean`
Default Value: `nil`
T means use tabs when performing computed indentations

:indentation-chars OPTION
Type: `stringp`
Default Value: `" "`
List of characters to skip over during autoindent.

:iparse-debug OPTION
Type: `boolean`
Default Value: `nil`

Turn on incremental parser tracing.

:killring-max-size OPTION
Type: `fixp`
Default Value: `16`
Maximum number of kills retained in kill ring.

:kills-to-clipboard OPTION
Type: `boolean`
Default Value: `nil`
If true, the clipboard is used instead of the local kill ring in kill commands.

:ls-flags OPTION
Type: `stringp`
Default Value: `"-lF"`
Flags for /bin/ls command use in List-Files

:minor-mode OPTION
Type: `stringp`
Default Value: `"Normal"`
Name for local binding set.

:mode-line-fmt OPTION
Type: `stringp`
Default Value: `"File %F %* %C %W Mode:  %M"`
Format string for status information.

:pan-load-search-path OPTION
Type: `listp`
Default Value: `(|.| " "piper/lib/pan)`
Search path used by Load-File command.

:pause-ticks OPTION
Type: `fixp`
Default Value: 250
Constant multiplier for Pause command.

:proportional-scroll OPTION
Type: `boolean`
Default Value: `t`
Vertical scroll proportionally, else fixed screenful at a time.

:region-highlight-op OPTION
Type: `(:underline :invert)`
Default Value: `:underline`
Operation for highlighting: :underline or :invert.

:row-popup OPTION
Type: `fixp`
Default Value: `4`
Minimum number rows to move window when scrolling up/down.

:save-on-system-error OPTION
Type: `boolean`
Default Value: `nil`
Save all buffers when error is encountered?

:show-node-reuse OPTION
Type: `boolean`
Default Value: `nil`
Turn on tracing for incremental parsing node reuse

:syntax-classes OPTION
Type: `listp`
Default Value: `(:word-char :space-char :punct-char :lbracket :rbracket :other)`
List of valid syntax classes.

:tabwidth OPTION
Type: `fixp`
Default Value: 8
Number of characters per tab.

:text-font-map OPTION
Type: `listp`
Default Value: `("screen.r.12" "screen.b.12" "serif.r.12" "cour.r.12" "cour.b.12")`
Font map for text viewers.

:verbose-load OPTION
Type: `boolean`
Default Value: `nil`
T iff verbose file loading is desired.

:verbose-parse OPTION
Type: `boolean`
Default Value: `nil`
T iff verbose parsing information is desired.

:verbose-trace OPTION
Type: `boolean`
Default Value: `nil`
T iff verbose tracing mode is desired.

:visible-flags OPTION
Type: `check-flags`
Default Value: `(:text-modified :read-only)`
Flags visible on control panel.

:win-cols OPTION
Type: `fixp`
Default Value: 78
Default number of columns per window.

:win-cols-min OPTION
Type: `fixp`
Default Value: 10

Minimum number of columns in window; guards against user resizing.

:win-rows OPTION
Type: `fixp`
Default Value: 30
Default number of rows per window.

:win-rows-min OPTION
Type: `fixp`
Default Value: 2
Minimum number of rows in window; guards against user resizing.

:wrap-@level OPTION
Type: `boolean`
Default Value: `t`
Automatically wrap operand hierarchy from top to bottom, or from bottom to top.

:zero-index-lines OPTION
Type: `boolean`
Default Value: `t`
If T, then arguments to Goto-Line are interpreted as 0-indexed. Otherwise, the arguments are interpreted as 1-indexed.

F Flags Defined in *Pan*

:auto-exec FLAG
Presentation: `"E"`
Behaviour: `:invisible-when-cleared`
Set when auto-execution related to file suffix has occurred.

:blank-flag FLAG
Presentation: `" "`
Behaviour: `:invisible-when-cleared`
Dummy flag for padding flag array.

:lex-ok FLAG
Presentation: `"L"`
Behaviour: `:gray-when-cleared`
Set when language file has been scanned.

:panic FLAG
Presentation: `"P"`
Behaviour: `:invisible-when-cleared`
Set when panic-mode error recovery has been invoked.

:parse-errors FLAG
Presentation: `"!"`
Behaviour: `:invisible-when-cleared`
Set when there are errors in the parse tree.

:re-search-successful FLAG
Presentation: `"9"`
Behaviour: `:invisible-when-cleared`

Set if the last regular expression search was was successful

:read-only FLAG
Presentation: "$"
Behaviour: `:invisible-when-cleared`
Set when buffer file cannot be written.

:text-modified FLAG
Presentation: "*"
Behaviour: `:invisible-when-cleared`
Set when buffer text has been modified but not saved.

:tree-ok FLAG
Presentation: "T"
Behaviour: `:gray-when-cleared`
Set when language file has been parsed.

G *Pan* for GNU Emacs Users

Since *Pan*'s text-oriented facilities are modeled on the *emacs* family of text editors, users familiar with *emacs* will find much that is familiar. This appendix will help *emacs* users get started with *Pan* as conveniently as possible.

This appendix compares *Pan* version 1.9 and the *emacs* editor in use locally, GNU Emacs Version 18. The comparison reveals compatibility between the two *as text editors*, but does not pretend to give a comprehensive picture of the functionality of either system. Consult the main body of this manual for a more thorough introduction to *Pan*.

Pan also provides for manipulating and editing programs using the syntax and semantics of the language being edited. This structure-based, description-driven facility is fundamentally more powerful than the language modes supported by *emacs*, but is not discussed in this appendix.

Finally, *Pan* is extensible and customizable in the spirit of *emacs*, but techniques for doing so are beyond the scope of this appendix.

G.1 Key Bindings

Both *Pan* and GNU Emacs use *keymaps*, a dynamic mechanism for binding editor commands to keystrokes (or keystroke sequences). A *global keymap* is always present during an editing session, but may be effectively extended and altered by a *local keymap* associated with each buffer being edited. Local keymaps are typically created as part of special editing modes (see G.8, "Special Editing Modes").

This section compares only *default* key bindings in the global keymap; these are the bindings normally in effect for ordinary text editing. *Keymaps*, both global and local, may be easily customized in both editors; both editors support many commands that are not bound in the default keymap.

The key bindings currently in effect for a *Pan* buffer may be displayed by invoking the "Key Bindings" command from the help menu.

Columns 1 and 2 of this list are derived from the global default key bindings in the local version of GNU Emacs. Column 3 identifies *Pan*'s compatibility with one of the following symbols:

●	some equivalent supported now
o	some equivalent anticipated, but unimplemented
!	now bound to something else in *Pan*, as noted
blank	no equivalent supported in *Pan*, contributions welcome

and column 4 lists the *Pan* command binding. Many potential key stroke sequences remain unbound in both editors, and there are many commands to which no sequences are bound. A list of all available *Pan* commands may be displayed by invoking the "Commands" command from the help menu.

Key	GNU Emacs Binding	Code	*Pan* Binding
C-@	set-mark-command	●	Set-Mark
C-a	beginning-of-line	●	Move-To-BOL
C-b	backward-char	●	Previous-@Level
C-c	mode-specific-command-prefix	●	Command prefix
C-d	delete-char	●	Delete-@Level
C-e	end-of-line	●	Move-To-EOL
C-f	forward-char	●	Next-@Level
C-g	keyboard-quit	●	Abort-Command
C-h	help-command	!	Delete-Previous-Character
TAB	indent-for-tab-command	●	Self-Insert
LFD	newline-and-indent	●	Newline-And-Indent
C-k	kill-line	●	Kill-To-EOL
C-l	recenter	!	Redraw
RET	newline	●	Insert-Newline
C-n	next-line	●	Next-Line
C-o	open-line	●	Open-Line
C-p	previous-line	●	Previous-Line
C-q	quoted-insert	●	Quote-Insert
C-r	isearch-backward	●	Re-Search-Backward
C-s	isearch-forward	●	Re-Search-Forward
C-t	transpose-chars	●	Transpose-Characters
C-u	universal-argument	●	Read-Prefix-Arguments
C-v	scroll-up	●	Forward-Vscroll
C-w	kill-region	●	Kill-Selected-Region
C-x	Control-X-prefix	●	Command prefix
C-y	yank	●	Yank-From-Kill-Ring
C-z	suspend-emacs	!	Command prefix
ESC	ESC-prefix	●	Command prefix
C-]	abort-recursive-edit		
C-_	undo		
SPC .. ~	self-insert-command	●	Self-Insert
DEL	delete-backward-char	●	Delete-Previous-Character
C-x C-a	add-mode-abbrev	!	Toggle-Auto-Fill
C-x C-b	list-buffers	●	Show-Buffers
C-x C-c	save-buffers-kill-emacs	●	Write-Files-Exit
C-x C-d	list-directory	●	List-Files
C-x C-e	eval-last-sexp		
C-x C-f	find-file	●	Visit-File
C-x C-h	inverse-add-mode-abbrev		
C-x TAB	indent-rigidly	!	Insert-File
C-x LFD			
C-x C-k			
C-x C-l	downcase-region [see Esc C-l]	!	Load-file
C-x RET		!	Save-All-buffers
C-x C-n	set-goal-column		

Key	GNU Emacs Binding	Code	*Pan* Binding
C-x C-o	delete-blank-lines	●	Delete-Blank-Lines
C-x C-p	mark-page		
C-x C-q	toggle-read-only	●	Toggle-Read-Only
C-x C-r	find-file-read-only		
C-x C-s	save-buffer	●	Save-Buffer-File
C-x C-t	transpose-lines	●	Transpose-Lines
C-x C-u	upcase-region [see Esc C-u]		
C-x C-v	find-alternate-file		
C-x C-w	write-file	●	Write-To-File
C-x C-x	exchange-point-and-mark	●	Swap-Dot-And-Mark
C-x C-y			
C-x C-z	suspend-emacs	!	Interrupt-Pan
C-x ESC	repeat-complex-command		
C-x !		!	Cycle-Show-Kill
C-x $	set-selective-display		
C-x (	start-kbd-macro		
C-x)	end-kbd-macro		
C-x +	add-global-abbrev		
C-x -	inverse-add-global-abbrev		
C-x .	set-fill-prefix		
C-x /	point-to-register		
C-x 0	delete-window	●	Close-Active-Viewer
C-x 1	delete-other-windows		
C-x 2	split-window-vertically	●	Open-Another-Viewer
C-x 4	ctl-x-4-prefix		
C-x 5	split-window-horizontally		
C-x ;	set-comment-column		
C-x <	scroll-left	●	Left-Hscroll
C-x =	what-cursor-position	●	Show-Status-Line
C-x >	scroll-right	●	Right-Hscroll
C-x ?		!	Show-Kill
C-x [	backward-page		
C-x]	forward-page		
C-x ^	enlarge-window	o	
C-x ‘	next-error		
C-x a	append-to-buffer		
C-x b	switch-to-buffer	●	Visit-Buffer
C-x d	dired [see C-x C-d]		
C-x e	call-last-kbd-macro		
C-x f	set-fill-column	●	Set-Auto-Fill-Column
C-x g	insert-register		
C-x h	mark-whole-buffer	●	Select-Buffer
C-x i	insert-file [see C-x C-i]		
C-x j	register-to-point		
C-x k	kill-buffer	●	Remove-Buffer

Key	GNU Emacs Binding	Code	*Pan* Binding
C-x l	count-lines-page	!	Goto-Line
C-x m	mail		
C-x n	narrow-to-region		
C-x o	other-window		
C-x p	narrow-to-page		
C-x q	kbd-macro-query		
C-x r	copy-rectangle-to-register		
C-x s	save-some-buffers [see C-x C-m]		
C-x u	advertised-undo	●	Undo
C-x w	widen		
C-x x	copy-to-register		
C-x y			
C-x {	shrink-window-horizontally		
C-x }	enlarge-window-horizontally		
C-x DEL	backward-kill-sentence	o	
ESC C-@	mark-sexp	●	Select-Expr
ESC C-a	beginning-of-defun	●	[see lisp-mode]
ESC C-b	backward-sexp	●	Backward-Expr
ESC C-c	exit-recursive-edit	!	Capitalize-Selection
ESC C-d	down-list	!	Deselect-Region
ESC C-e	end-of-defun	●	[see lisp-mode]
ESC C-f	forward-sexp	●	Forward-Expr
ESC C-h	mark-defun	●	[see lisp-mode]
ESC TAB		!	Indent-Like-Previous-Line
ESC LFD	indent-new-comment-line		
ESC C-k	kill-sexp	●	Kill-Expr
ESC C-l		!	Lowercase-Selection
ESC RET		!	Move-Selection-To-Cursor
ESC C-n	forward-list		
ESC C-o	split-line	●	Split-Line
ESC C-p	backward-list		
ESC C-q			
ESC C-r			
ESC C-s	isearch-forward-regexp		
ESC C-t	transpose-sexps		
ESC C-u	backward-up-list	!	Uppercase-Selection
ESC C-v	scroll-other-window		
ESC C-w	append-next-kill	!	Select-Region-Dot-To-Mark
ESC C-y		!	Copy-Selection-To-Cursor
ESC ESC	eval-expression	●	Execute-Lisp-Line
ESC C-\	indent-region		
ESC SPC	just-one-space	●	Just-One-Space
ESC !	shell-command	!	Cycle-Show-Clipboard
ESC $	spell-word		
ESC %	query-replace	!	Show-Match

Key	GNU Emacs Binding	Code	*Pan* Binding
ESC '	abbrev-prefix-mark		
ESC (	insert-parentheses	●	Insert-Parentheses
ESC)	move-past-close-and-reindent		
ESC ,	tags-loop-continue		
ESC -	indent-for-comment		
ESC .	find-tag		
ESC 0..9	digit-argument		
ESC ;	indent-for-comment		
ESC <	beginning-of-buffer	●	Move-To-BOB
ESC =	count-lines-region	o	
ESC >	end-of-buffer	●	Move-To-EOB
ESC ?		!	Show-Clipboard
ESC @	mark-word	●	Select-Word
ESC [	backward-paragraph	o	
ESC \	delete-horizontal-space	●	Delete-Horizontal-Space
ESC]	forward-paragraph	o	
ESC ^	delete-indentation	●	Delete-Indentation
ESC a	backward-sentence	o	
ESC b	backward-word	●	Previous-Word
ESC c	capitalize-word	●	Capitalize-Word
ESC d	kill-word	●	Kill-Word
ESC e	*forward-sentence*	o	
ESC f	forward-word	●	Next-Word
ESC g	fill-region		
ESC h	mark-paragraph		
ESC i	tab-to-tab-stop		
ESC j	indent-new-comment-line		
ESC k	kill-sentence	o	
ESC l	downcase-word	●	Lowercase-Word
ESC m	back-to-indentation	●	First-Non-Blank
ESC q	fill-paragraph		
ESC r	move-to-window-line		
ESC t	transpose-words	o	
ESC u	upcase-word	●	Uppercase-Word
ESC v	scroll-down	●	Backward-Vscroll
ESC w	copy-region-as-kill	●	Copy-Selection-As-Kill
ESC x	execute-extended-command	●	Execute-Named-Command
ESC y	yank-pop	●	Cycle-Yank
ESC z	zap-to-char		
ESC !	shell-command-on-region		
ESC ~	not-modified		
ESC DEL	backward-kill-word	●	Kill-Previous-Word

The auxiliary table below displays *Pan* default bindings for keys (and key sequences) that do not exist in a strictly keyboard based editor like GNU Emacs.

Key	*Pan* Binding
Mouse_Left	Mouse-Select-@Level
Mouse_Middle	Mouse-Extend-@Level
Mouse_Right	Execute-From-Menu
Esc-Mouse_Left	Mouse-Select-Fullword
Esc-Mouse_Middle	Mouse-Extend-Selection-Fullword
L4	Undo
L6	Copy-To-Clipboard
L8	Paste-From-Clipboard
Esc-L8	Replace-From-Clipboard
L10	Cut-To-Clipboard
F1	Set-@Level-To-Character
F2	Up-@Level
F3	Down-@Level

Section 2.3.1 in the manual discusses *Pan* key binding more generally; section 4.2 explains how to customize key bindings.

G.2 Menus

Commands in *Pan* may be bound to menus as conveniently as they may be bound to keys, both locally and globally. Menu bindings have the advantage of being visible to the user, but the disadvantage of taking longer to invoke.

Default menu bindings include some commands that are bound to both menus (for easy learning) and to keys (for convenience). Notable examples are the clipboard operations (see G.5, "Cut/Paste/Kill/Yank").

The base buffer and help buffer each have specialized menus. All other viewers provide a standard menu to which special purpose items may be appended in special editing modes (see G.8, "Special Editing Modes").

The *Pan* menu associated with a viewer appears in response to a press on the right mouse button when the cursor is over the viewer's text viewing area.

Section 2.3.2 in the manual discusses *Pan* menu binding more generally; section 4.2 explains how to customize menu bindings.

G.3 Undo

Pan has a general *undo* facility that behaves more like the one in *vi* than the one in GNU Emacs. The command Undo (bound by default to keys L4 and ^X-u and to the main menu) reverses the most recent editing action. Undo is itself an editing action, so two consecutive invocations of Undo will result in no net change.

Certain classes of editing actions (notably insertions and deletions), when performed consecutively, are treated by *Pan*'s Undo as a single action that represents their aggregate effect.

Unlike GNU Emacs, cursor motion in *Pan* is considered an undoable editing action. Section 3.5 in the manual discusses *Pan*'s undo mechanism more generally.

G.4 Operand Level

The *operand level* mechanism in *Pan* has no counterpart in GNU Emacs or any other common editor. The operand level (or simply the *level*) is a persistent mode, local to each viewer. Its current value (usually one of "character," "word," or "line") is visible on the viewer's control panel and may be set either from the control panel or by three commands bound to keys F1, F2, and F3.

The level specifies how the operands of certain generic commands (those whose names contain *@Level*) will be determined. For example, the generic command Next-@Level moves the cursor forward by one character when the level is "character", but when the level is "word" it moves the cursor forward by one word. Level-sensitive (generic) commands are bound by default so that when the level is "character," they mimic GNU Emacs. Thus, Next-@Level is bound to ^F, the slot where the Next-Character command would otherwise appear.

The motivation for this mechanism becomes clear in special editing modes where tree building occurs. Extra operand levels may be defined by specifications in the *Ladle* description of a programming language. For example, in asple-mode (asple is a demonstration programming language) possible levels are:

Declaration
Statement
Expression
Error
Lexeme
Line
Word
Character

The operand level mechanism is supported by a second level of mapping, analogous to keymaps. Thus, for example, the key ^F is bound to Next-@Level in the keymap; Next-@Level is bound, in turn, to the command Next-Character when the level is "character." Operand bindings are, of course, customizable too.

Section 2.3.3 in the manual discusses *Pan*'s operand binding mechanism more generally.

G.5 Cut/Paste/Kill/Yank

The functional area of *Pan* most likely to confuse experienced GNU Emacs users is the management of text that is to be deleted and/or moved among buffers. The two editors support models that are superficially alike but differ in crucial ways. The potential for confusion is exacerbated by the inherently invisible nature of some manipulations in this category.

This section presents the basic model supported by each editor. It enumerates the basic abstractions (drawing the pivotal distinction between buffer-local and editor-global) and gives examples of commands that operate upon them. Section 3 of the manual presents a more thorough introduction to *Pan*'s editing model.

G.5.1 The GNU Emacs Model

Window Abstractions

- A GNU Emacs buffer may have one or more *windows* that provide an independently scrollable view of the buffer.
- Exactly one GNU Emacs window is *active* at any time.
- The *active* window has a visible *dot* (a.k.a. cursor), at a specific point in the buffer; the dot is constrained to be *always visible* in the active window.
- A visible, inactive window does not display its dot, but retains its location should it become active again. Invisible windows do not exist, and therefore retain no state.

Buffer Abstractions

- Each buffer may have a invisible *mark*, independent of any dots, at a specific point in the buffer.
- If a buffer has a mark currently set, the text interval between dot and mark in the acative window implicitly (and invisibly) defines the *region*.

Global Abstractions

- The editor contains a single, invisible *kill ring* onto which text from various buffers may be pushed.

Operations

- Killing any text (^K, ^W, etc.) pushes it onto the global kill ring.
- The command copy-region-as-kill (Esc-w) pushes the region onto the global kill ring without killing.
- The command yank (^Y) inserts the most recently pushed text (from any buffer) at the dot in a specific buffer.
- The command yank-pop (Esc-y) removes the result of an immediately preceding yank operation, pops the most recently pushed text off of the kill ring, and replaces the removed text with the new top of the kill ring.

There are, of course, more operations for manipulating the kill ring explicitly, but this description is sufficient for comparison. GNU Emacs also supports a number of named global *registers* into which text can be stored. *Pan* supports nothing similar, so they will not be discussed here.

G.5.2 The Pan Model

In *Pan* it is necessary to distinguish between *buffers* and *viewers*. A single file is always edited in a single buffer. A *Pan* buffer may have one or more viewers visible on the screen, or it may have none at all. Viewers are something like GNU Emacs windows, since many of them may be attached to a buffer, but they retain more state, more persistently than GNU Emacs windows.

Viewer Abstractions

- Each viewer associated with a buffer has a visible *dot* (a.k.a. cursor) at a specific point in the file. Like GNU Emacs, there as many dots associated with a buffer as there are viewers. Unlike GNU Emacs, (a) the dot is *not* constrained to be in the visible part of a viewer, (b) the dot is visible even when a viewer is not active (doesn't have the keyboard focus in SunView terminology), and (c) the location of the dot (along with size, scroll position, and screen location) persists while a viewer is invisible.

Buffer Abstractions

- Like GNU Emacs, each buffer may have an invisible *mark*, independent of viewer dots, at a specific point in the file.
- Somewhat like GNU Emacs, the text interval between a dot (in a viewer) and mark (in a buffer, if set) implicitly (and invisibly) defines the *region*. Unlike GNU Emacs, this region is of little interest to the *Pan* user.
- Each buffer may have a visible *selection*, independent of the mark and any dots. The selection appears highlighted wherever visible in any active viewer. The selection appears underlined, but a user option (global or buffer local) may be changed to request inverse highlighting instead.
- Each buffer has a *kill ring* that is, unlike GNU Emacs, local. The mechanisms by which successive kills are coalesced are less well developed than in GNU Emacs.

Global Abstractions

- The editor contains a single *clipboard*, also a ring, onto which text from various buffers may be pushed.

Operations

This section lists examples of *Pan* commands related to these abstractions. Each command mentioned will be followed by the key to which it is bound by default, if any.

- Like GNU Emacs, deleting text removes it from a buffer, and it is not retained (except on the undo stack):

Delete-Character (^D)
Delete-Previous-Character (DEL)
Delete-Word
Delete-Region-Dot-To-Mark
Delete-Selected-Region
Delete-Blank-Lines (^X-^O)
Delete-Horizontal-Space (Esc-)
etc.

- Killing any text pushes it onto the buffer's local kill ring:

  Kill-Word (Esc-d)
  Kill-To-EOL (^K)
  Kill-Selected-Region (^W)
  Kill-Region-Dot-To-Mark
  etc.

- Like GNU Emacs, text can be pushed onto the buffer's local kill ring without removing it:

  Copy-Selection-As-Kill (Esc-w)

- Like GNU Emacs, text may be retrieved from the top of a buffer's local kill ring by *yanking*, causing it to be inserted at the dot; Cycle-Yank specifies that the kill ring be cycled before the yank and insertion. However, unlike GNU Emacs, both yank operations may be modified by a prefix argument, in which case yanked text *replaces* the current selection instead of being inserted at the dot. Thus, *Pan*'s Yank-From-Kill-Ring with no prefix arguments (key sequence "^Y") is similar to the GNU Emacs ''yank'' command, and *Pan*'s Cycle-Yank *with* prefix arguments (key sequence "^U Esc-y") has an effect similar to the GNU Emacs ''yank-pop'' command.

  Yank-From-Kill-Ring (^Y)
  Cycle-Yank (Esc-y)

- A buffer's local kill ring may be cycled, bringing successive entries to the top, and the text currently at the top may be viewed in the Help buffer.

  Cycle-Kill
  Show-Kill (^X-?)
  Cycle-Show-Kill (^X-!)

- Some operations use the buffer's current selection as an operand (see also clipboard operations below):

  Append-Selection-To-File
  Apropos-Selection
  Write-Selection-To-File
  Kill-Selected-Region (^W)
  Lowercase-Selection (^X-^U)
  etc.

- One can create a selection either with the mouse or from the dot and mark; the latter option is included only for compatibility with emacs.

  Select-Region-Dot-To-Mouse (Mouse-Right)
  Select-Region-Dot-To-Mark (Esc-^W)
  Deselect-Region (Esc-^D)

- Some commands operate on both the selection and the dot. GNU Emacs has no equivalents because it supports no abstraction corresponding to a selection that is independent of the dot.

  Copy-Selection-To-Cursor (Esc-^Y)
  Move-Selection-To-Cursor (Esc-^M)

- Text may be pushed onto the clipboard from the current selection (Copy or Cut). Text may be retrieved from the top of the clipboard, and it may be either inserted at the dot (Paste) or used to replace the current selection (Replace).

  Copy-To-Clipboard (L6)
  Cut-To-Clipboard (L10)
  Paste-From-Clipboard (L8)
  Replace-From-Clipboard (Esc-L8)

- The clipboard may be cycled, bringing successive entries to the top, and the text currently at the top may be viewed in the Help buffer.

  Cycle-Clipboard
  Show-Clipboard (Esc-?)
  Cycle-Show-Clipboard (Esc-!)

G.6 Options

Many aspects of *Pan*'s operation may be controlled by the settings of options, somewhat analogous to GNU Emacs variables (although *Pan* also has variables). As with bindings, options may have both global and buffer-local values. The help command List-All-Options lists the option values currently in effect for the active buffer. Some user-level commands are available for setting options dynamically, for example Set-Auto-Fill-Column and Toggle-Read-Only. Section 4.3 of the manual explains in more detail how to manage *Pan* options.

G.7 Help

Pan has a help mechanism that is somewhat different from the one in GNU Emacs. All output from help commands appears in the distinguished buffer named "Help Info." Output from some commands (for example the command Describe that provides additional information about a selected *Pan* symbol) is inserted among existing text in the help buffer. The command Reset clears the contents of the help buffer; some help commands do this automatically before responding.

Section 3.6 in the manual presents a more detailed explanation of *Pan*'s help mechanism.

G.8 Special Editing Modes

Both editors support specialized *editing modes* that are optimized for editing certain classes of files, C programs or TeX documents for example. A mode is typically invoked by pattern matching against file extensions.

In its simplest form, a mode contains alternate key bindings and special purpose commands.

Pan modes can operate like GNU Emacs modes, but have additional flexibility. For example, *Pan* modes may also add special purpose submenus. Language editing modes load *Ladle* tables for parsing and tree building and provide extra commands for tree navigation and manipulation.

The short tables below describe two textual modes supported by GNU Emacs, along with such textual *Pan* equivalents as currently exist. These modes are experimental and incomplete. Full structure- and semantic-based *Pan* support for these languages is under development. For a discussion of *Pan*'s true language editing modes, see section 3.12 of the manual.

Lisp Mode Bindings

Pan contains a small set of text-oriented functions for operating on LISP expressions. More powerful functions await true language-based editing on LISP.

Key	GNU Emacs Binding	Code *Pan* Binding	
TAB	lisp-indent-line		
DEL	backward-delete-char-untabify		
ESC	Prefix Command		
ESC C-a	beginning-of-defun [std.]	●	Previous-Function
ESC C-e	end-of-defun [std.]	●	End-Of-Function
ESC C-h	mark-defun [std.]	●	Select-Function
ESC C-q	indent-sexp		
ESC C-x	lisp-send-defun		
ESC [	backward-paragraph [std.]	!	Previous-Function
ESC]	forward-paragraph [std.]	!	Next-Function

C Mode Bindings

There is no special mode yet for C in *Pan*, either text-oriented or language-based.

Key	GNU Emacs Binding	Code *Pan* Binding
TAB	c-indent-command	
DEL	backward-delete-char-untabify	
ESC	Prefix Command	
:	electric-c-terminator	
;	electric-c-semi	
}	electric-c-brace	
{	electric-c-brace	
ESC C-q	indent-c-exp	
ESC C-h	mark-c-function	